

RESOLUTION STRATEGIES FOR SERVERLESS COMPUTING IN INFORMATION CENTRIC NETWORKING

Inauguraldissertation

zur

Erlangung der Würde eines Doktors der Philosophie

vorgelegt der

Philosophisch-Naturwissenschaftlichen Fakultät

der Universität Basel

von

Christopher Max Constantin Scherb

aus Deutschland

Basel, 2020

Originaldokument gespeichert auf dem Dokumentenserver
der Universität Basel

edoc.unibas.ch



Dieses Werk ist unter dem Vertrag "Nicht kommerziell - Keine Bearbeitungen 4.0 International
(CC BY-NC-ND 4.0)" lizenziert. Die vollständige Lizenz kann unter

creativecommons.org/licenses/by-nc-nd/4.0/

eingesehen werden.

Genehmigt von der Philosophisch-Naturwissenschaftlichen Fakultät
auf Antrag von

Prof. Dr. Christian Tschudin, Fakultätsverantwortlicher, Dissertationsleiter
Prof. Dr. Jörg Ott, Korreferent

Basel, den 21.04.2020

Prof. Dr. Martin Spiess, Dekan

Dedicated to my family

The Internet is becoming the town square for the global village of tomorrow.

— Bill Gates

And it's interesting, when you look at the predictions made during the peak of the boom in the 1990s, about e-commerce, or internet traffic, or broadband adoption, or internet advertising, they were all right - they were just wrong in time.

— Chris Anderson

Zusammenfassung

Named Function Networking (NFN) ermöglicht es, Berechnungen im Rahmen von Information Centric Networking (ICN) durchzuführen. ICN kann Abfragen nach Daten ortsunabhängig, ohne Angabe des Rechners, der sie speichert, beantworten. NFN erweitert ICN um die Produktion von verarbeiteten Daten (Ergebnisse von Berechnungen), ohne Angabe, wo die Berechnung ausgeführt wird. Dabei wird der Berechnungsablauf mit Hilfe des λ -Calculus in Interest-Messages (IMs) kodiert. Basierend auf dieser Berechnungsdefinition verteilt das NFN die Berechnungen im Netzwerk und findet so für diese einen geeigneten Ausführungsort. Je nach spezifischer Verwendung des Netzwerks wird eine Entscheidung getroffen, wo die Berechnung durchgeführt werden soll: Eine *Lösungsstrategie*, die auf jedem einzelnen Knoten ausgeführt wird, legt fest, ob eine Berechnung weitergeleitet, in Teilberechnungen aufgespalten oder lokal ausgeführt werden soll.

Der Fokus dieser Arbeit liegt auf dem Entwerfen von Lösungsstrategien für bestimmte Szenarien und der Erstellung von "Ausführungsplänen", die auf dem Zustand des Netzwerkes und vorherigen Entscheidungen basieren. Dabei beginnen wir mit einer einfachen, für Datenzentren geeigneten, Lösungsstrategie und konzentrieren uns darauf, die Verteilung der Last innerhalb eines oder sogar zwischen mehreren Datenzentren zu verbessern. Die von uns entwickelten Lösungsstrategien berücksichtigen die Größe der Eingabedaten sowie die Last auf den Knoten, sodass wir nach Ausführkosten bewertete Ausführungspläne erhalten. Aus diesen können dann die günstigsten gewählt werden.

Zusätzlich verwenden wir diese Pläne, um Ausführungsvorlagen zu erstellen. Für eine spezifische Anwendung wird eine Lösungsstrategie erstellt, in dem die Ausführung der spezifischen Anwendung simuliert wird. Außerdem haben wir Lösungsstrategien für *Edge Computing* entwickelt, die mobile Szenarien, wie zum Beispiel die Fahrzeugvernetzung, handhaben können. Diese *Mobile Edge Computing Lösungsstrategien* lösen das Problem der häufigen Verbindungswechsel zwischen Road-Side-Units. All diese Lösungsstrategien wurden mit einem Simulationssystem evaluiert und mit dem *state-of-the-art* Verhalten von Ausführungsumgebungen für Datenzentren verglichen. Für die Fahrzeugnetzwerkstrategie haben wir Road-Side-Units erweitert und unsere auf NFN basierende Lösungsstrategie implementiert, so dass wir unsere mobile edge Lösungen in einem realen Szenario testen und verifizieren konnten.

Abstract

Named Function Networking (NFN) offers to compute and deliver results of computations in the context of Information Centric Networking (ICN). While ICN offers data delivery without specifying the location where these data are stored, NFN offers the production of results without specifying where the actual computation is executed. In NFN, computation workflows are encoded in (ICN style) Interest Messages using the lambda calculus and based on these workflows, the network will distribute computations and find execution locations. Depending on the use case of the actual network, the decision where to execute a computation can be different: A *resolution strategy* running on each node decides if a computation should be forwarded, split into subcomputations or executed locally.

This work focuses on the design of *resolution strategies* for selected scenarios and the online derivation of "execution plans" based on network status and history. Starting with a simple *resolution strategy* suitable for data centers, we focus on improving load distribution within the data center or even between multiple data centers. We have designed *resolution strategies* that consider the size of input data and the load on nodes, leading to priced execution plans from which one can select the ones with the least costs.

Moreover, we use these plans to create *execution templates*: Templates can be used to create a *resolution strategy* by simulating the execution using the planning system, tailored to the specific use case at hand.

Finally we designed a *resolution strategy* for edge computing which is able to handle mobile scenarios typical for vehicular networking. This "mobile edge computing *resolution strategy*" handles the problem of frequent handovers to a sequence of road-side units without creating additional overhead for the non-mobile use case.

All these *resolution strategies* were evaluated using a simulation system and were compared to the state of the art behavior of data center execution environments and/or cloud configurations. In the case of the vehicular networking strategy, we enhanced existing road-side units and implemented our NFN-based system and plan derivation such that we were able to run and validate our solution in real world tests for mobile edge computing.

Acknowledgements

I would like to thank my supervisor Prof. Dr. Christian Tschudin for giving me the opportunity for this work, for being my adviser, for always giving hints and viewing problems from a different angle helping to find new solutions. Moreover, I would like to thank my co-referent Prof. Dr. Jörg Ott. A special thanks goes to Dr. Manolis Sifalakis who was a big help during the beginning of my PhD Studies.

I want to thank my colleague and friend Claudio Marxer for the discussions about the research and for helping me to find solutions to problems I faced as well as for the time we spent together at conferences and writing papers.

My thanks go to Dima Mansour and Dr. Urs Schnurrenberger for being my colleagues and for the work in lectures we have been assisting together.

I would like to thank Dennis Greve and Dr. Marco Wagner from Robert Bosch GmbH for the good cooperation in research and development of connected vehicles. Additionally I thank Cenk Gündogan and Peter Kietzmann for cooperation in the development of software for the Internet of Things.

Furthermore, I want to say Thank you to the students Claudio Marxer and Balazs Faludi for their professional work during their master thesis, which I supervised. The same thanks goes to my Bachelor Students Sebastian Lukas Phillip and Samuel Emde.

I would like to thank Prof. Dr. Dirk Kutscher for hints and ideas as well as Prof. Dr. Jeff Burke, Prof. Dr. Lixia Zhang, Prof. Dr. Alexander Afanasyev, Dr. Eve Schooler, Dr. Dave Oran, Prof. Dr. Thomas Schmitt and Prof. Dr. Matthias Wählisch.

Last but not least, I would like say “Thank You” to my wife Mahnaz Parian-Scherb for supporting me during this thesis and for advises and hints. I also want to thank my brother Sebastian Scherb, my sisters Christiane Scherb and Viktoria Scherb as well as my parents Heide von Massow-Scherb and Dr. Michael Scherb for proof-reading my thesis and their support in general.

Without you, this thesis would not exist.

Contents

Zusammenfassung	ix
Abstract	xi
Acknowledgements	xiii
List of Figures	xxi
List of Tables	xxv
List of Algorithms	xxvii
List of Acronyms	xxix
I Introduction, Background & Related Work	1
1 Introduction	3
1.1 A History of Computer Networking and the Internet	4
1.2 Data Centric Approaches on Network Layer	5
1.3 Computation in Networks	6
1.4 Contributions	7
2 Background & Related Work	9
2.1 Information Centric Networking (ICN)	9
2.1.1 The Content Centric Networking (CCN) Forwarder	10
2.1.2 CCN Forwarding	12
2.1.3 Aging of the CCN Data Structures	13
2.1.4 Chunking	14
2.2 Named Function Networking (NFN)	15
2.2.1 Encoding Computations in CCN names using the λ calculus	16
2.2.2 Resolution Strategies and Pinned Functions in Named Function Networking (NFN)	19
2.2.3 Timeout Prevention in NFN	21
2.3 Computing in Information Centric Networking (ICN) Networks .	24
2.3.1 Service Centric Networking	25

2.3.2	Named Functions as a Service	25
2.3.3	Remote Method Invocation in ICN	26
2.3.4	Compute First Networking	26
2.3.5	Intel: Joint Optimization of Routing and Caching in Heterogeneous Wireless Networks	26
2.4	Database Query Optimization	27
2.5	Serverless Computing	27
2.5.1	Existing Research in Serverless Computing	28
2.5.2	Use Cases of Serverless Computing	29
2.6	Edge/Fog Computing	29
2.7	Vehicle-To-Everything Communication and the Electronic Horizon	33
2.7.1	Road Side Units (RSUs)	33
2.7.2	Vehicle-to-Everything Communication	34
2.7.3	Electronic Horizon	35
II	Advanced Resolution Strategies in Named Function Networking	37
3	Information Used for Resolution Strategies in Named Function Networking	39
3.1	Information used for Resolution Strategies	41
3.2	NFN Resolution Strategies	43
4	NFN Map-Reduce Resolution Strategy	45
4.1	Analyzing the NFN Expression using the Abstract Syntax Tree (AST)	46
4.2	Resolution Based on the Analysis of the Abstract Syntax Tree (AST)	48
5	NFN Mobile-Edge-Computing Resolution Strategy	53
5.1	Offloading Computations in Static Scenarios	54
5.2	Computation Offloading Strategies for Mobile Scenarios	55
5.3	NFN Resolution Strategies for Computation Offloading in Mobile Scenarios	57
5.4	Uploading Data for Mobile Edge Computing with NFN	60
5.4.1	Name Space for Edge Computing in NFN	61
5.4.2	Data Upload using Meta Data Objects	61
5.5	Requirements to Combine Resolution Strategies	62
5.6	Optional Parameters in NFN Computations	64

6	NFN Plan-Based Resolution Strategy	67
6.1	Plans for Executing NFN Computation	68
6.2	Local Named Plans and Caching/Reusing of Plans	70
6.3	Requesting Meta Information and Creating Plans	72
6.3.1	Metrics to Compute Optimal Execution Location	73
6.3.2	User Defined Metric	74
6.3.3	Requesting Information about the Network Status	75
6.3.4	Creating Plans	77
6.3.5	Clustering Regions and Prefixes	80
6.4	Reusing of Plans	83
6.5	Presharing of Information Required to Create Plans	85
6.5.1	Transporting Data to the Planning Unit	87
7	NFN Template-Based Resolution Strategy	89
7.1	Creating Templates	90
7.1.1	Templates based on the Knowledge of the Network Topology	91
7.1.2	Templates based on History or Simulation	92
7.2	Forwarding on Templates	93
III	Application Scenarios & Evaluation	95
8	Application Scenarios for NFN Resolution Strategies	97
8.1	Applying NFN Resolution to (Serverless) Cloud Computing	98
8.1.1	To-Data-First Application	100
8.1.2	Map-Reduce optimized Application	100
8.1.3	NFN and Amazon Lambda/Serverless Computing	102
8.2	Applying NFN Resolution to Edge Computing	103
8.3	Applying NFN Resolution to Mobile Computing	105
8.4	Applying NFN Resolution to Multitier Computing	106
9	Evaluation	111
9.1	Software & Implementation	112
9.2	Data Center Evaluation	112
9.2.1	Test Scenario and Configuration:	113
9.2.2	Baseline: Evaluation of the To-Data-First Resolution Strategy	117
9.2.3	Evaluation of the Map-Reduce Resolution Strategy	118
9.2.4	Evaluation of the Plan-Based Resolution Strategy	121
9.2.5	Evaluation of the Template-Based Resolution Strategy . . .	124

9.2.6	Efficiency of Templates	129
9.2.7	Finding a good Metric for Plan-/Template-Based Resolution Strategies	130
9.3	Mobile Scenarios Evaluation	132
9.3.1	Test Scenario and Configuration	132
9.3.2	Baseline: Computation with the To-Data-First Resolution Strategy	134
9.3.3	Computation with the Resolution Strategy for Mobile Edge Computing	135
9.3.4	Computation with the Resolution Strategy for Mobile Edge Computing and Mobile Data Uploading	136
9.3.5	Comparison Cloud vs Edge Computing	137
9.3.6	Including Data from the Cloud and Fog/Multitier Computing	139
9.3.7	Other Experiments	140
9.3.8	Discussion of the Results	143
IV	Discussion, Conclusion & Future Work	147
10	Discussion	149
10.1	NFN Design and Privacy	149
10.2	Billing in NFN	150
10.3	Dependency between Routing and Resolution Strategies	151
10.4	Metrics for the Plan- and Template-based Resolution Strategies	152
10.5	Vehicular Computing and Hand-Overs	153
10.6	Analysis of the Results of the Evaluation	154
10.6.1	NFN and Cloud Computing	154
10.6.2	NFN and Mobile Edge Computing	155
10.6.3	NFN and Multitier Computing	156
10.6.4	Summary of the results	156
11	Conclusion	157
12	Future Work	159
12.1	Planning System	159
12.2	Edge Computing	159
12.3	Real World Experiments	160
12.4	Planning in Swarm Scenarios	161

V	Appendix	163
A	Raw Results of the Data Center Simulations	165
B	Percentage of Runtime Parts of the Data Center Simulations	175
	Bibliography	185

List of Figures

2.1	Scheme of a CCN Forwarder [JST ⁺ 09].	11
2.2	The forwarding process of CCN [ZAB ⁺ 14].	12
2.3	Scheme of an index structure for addressing chunks as it is used by File-like ICN Collection (FLIC) [TS14].	15
2.4	Rewriting and Remapping process in NFN.	19
2.5	Two cases of NFN execution.	20
2.6	Scheme of a NFN <i>resolution process</i>	22
2.7	Resolving links when receiving Thunks instead of the final result. . .	23
2.8	Keep Alive Messages to prevent Pending Interest Table (PIT) timeouts for computations.	25
2.9	Scheme of Cloud Computing and Edge Computing [Lea18].	31
2.10	Difference in the typical data flows in Cloud and Edge Computing. . .	32
2.11	RSUs have to hand over computations to match the movement of the vehicle.	34
2.12	Scheme of the Vehicle-to-Everything (V2X) communication (Graphic by Robert Bosch GmbH) [GWA ⁺ 17].	35
2.13	Scheme of the Electronic Horizon on a section of a road (Graphic by Robert Bosch GmbH) [GWA ⁺ 17].	36
3.1	Scheme of the resolution of a computation in NFN	40
3.2	Information Levels used for Resolving a Computation in NFN.	42
4.1	Basic Resolution of Computation in NFN (a) vs Resolution based on information out of the Forwarding Information Base (FIB) (b).	47
4.2	Example of an AST of a NFN expression.	48
4.3	Schematic representation of an AST as used for analyzing NFN ex- pressions.	49
4.4	Example for independent components within the AST for a NFN ex- pression	50
4.5	Schemes of the different cases of the resolution process using infor- mation from the FIB and the Content Store (CS).	51
5.1	Offloading a Computation into the Network.	56
5.2	Recovering data lost due to mobility from cache.	57

5.3	Mobility Challenge with Edge Computing.	58
5.4	Out of Range Problem with a mobile client using Edge Computing.	58
5.5	Computing can be faster than searching for a cached result.	59
5.6	Uploading data to a single base station may not be sufficient.	61
5.7	Name space for Mobile Edge Computing.	62
5.8	Uploading data over multiple base stations.	63
5.9	Process of fetching data from neighbored base stations.	63
5.10	Rules to prevent loops when using different <i>resolution strategies</i> in the same network	65
5.11	Example for Deadline Oriented Parameter Requests.	66
6.1	Scheme of Planning and Executing Plans	68
6.2	Scheme of Executing Plans and Subplans	70
6.3	Resolution with <i>Named Plans</i>	71
6.4	Example of NFN with <i>Named Plans</i>	72
6.5	Scheme of Creating Plans	73
6.6	Combining Subplans to an overall Plan	76
6.7	Finding the Best Combination of Forwarding and Local Computations	81
6.8	Clustering of Prefixes	82
6.9	Reusing of not exactly matching plans	84
6.10	Reusing of not exactly matching plans by comparing file sizes	85
6.11	Problem with diffuse approach and pre-distribution of data.	86
6.12	Pushing Data to the Planning Unit	88
7.1	Example for Template Matching	90
7.2	Example for Template Creation based on the Topology	91
7.3	Comparing Templates and Plans using a Matching Score	94
8.1	NFN for Serverless Computing: Data Center usage only vs exposing NFN to the client.	99
8.2	The <i>To-Data-First resolution strategy</i> within a data center.	101
8.3	The <i>To-Data-First resolution strategy</i> outside of the data center to achieve fast forwarding and the <i>Map-Reduce resolution strategy</i> within the data center for better data distribution.	102
8.4	The <i>Map-Reduce resolution strategy</i> outside of a data center to distribute subcomputations to the data center storing the input data.	103
8.5	Edge Computing Node and NFN	104
8.6	Road Side Unit	106
8.7	Car equipped with IEEE 802.11p communication system.	107

8.8	Scheme of Edge, Fog and Cloud Computing.	108
8.9	Edge Computing in Multitier scenarios.	109
8.10	Fog Computing in Multitier scenarios.	110
8.11	Cloud Computing in Multitier scenarios.	110
9.1	Simulated small data center used in the first test scenario.	113
9.2	Evaluation Scenario with two simulated data centers.	114
9.3	Simulated big data center used in the second test scenario.	115
9.4	Evaluation Scenario with five simulated data centers.	116
9.5	Comparing the Planning Time for both small and big data center simulations.	123
9.6	Comparing the Planning Time divided by the number of subcomputations for both small and big data center simulations.	124
9.7	Comparing the Planning Time for both small and big data center simulation with and without clustering.	125
9.8	Overview over the Simulation Results in the small data center scenario.	127
9.9	Overview over the Simulation Results in the big data center scenario.	128
9.10	Template Efficiency Simulation.	130
9.11	Comparing different Metrics.	131
9.12	Scheme of the testing setup for simulating edge computing.	133
9.13	Percentage of Delivered Results in a Simulation using the <i>To-Data-First resolution strategy</i> with two and eight cars.	135
9.14	Percentage of Delivered Results in a Simulation using the <i>Mobile-Edge-Computing resolution strategy</i> with two and eight cars.	136
9.15	Percentage of Delivered Results in a Simulation using the <i>Mobile-Edge-Computing resolution strategy</i> and Mobile Data Upload Strategy with two and eight cars.	137
9.16	Comparing Cloud vs Edge Computing for Mobile Scenarios.	138
9.17	Comparing Multitier Computing with computing the <i>/prcoess</i> function on Edge only and in the Cloud only.	140
9.18	Scheme of Computing in absence of Infrastructure.	145
9.19	Test setup for the real world Road Side Unit (RSU) experiment.	146
9.20	Test vehicle driving along a RSU.	146

List of Tables

3.1	Summary of Information Usage Levels	43
9.1	Number n of subcomputations/input video parts and overall video length	116
9.2	Summary of the Results of the Evaluation of the <i>To-Data-First resolution strategy</i> (Mean Value).	118
9.3	Summary of the Results of the Evaluation of the <i>To-Data-First resolution strategy</i> (Variance).	119
9.4	Summary of the Results of the Evaluation of the <i>Map-Reduce resolution strategy</i> (Mean Value).	120
9.5	Summary of the Results of the Evaluation of the <i>Map-Reduce resolution strategy</i> (Variance).	121
9.6	Summary of the Results of the Evaluation of the <i>Plan-Based resolution strategy</i> (Mean Value).	122
9.7	Summary of the Results of the Evaluation of the <i>Plan-Based resolution strategy</i> (Variance).	123
9.8	Summary of the Results of the Evaluation of the <i>Template-Based resolution strategy</i> (Mean Value).	126
9.9	Summary of the Results of the Evaluation of the <i>Template-Based resolution strategy</i> (Variance).	126
9.10	Simulated Speed and Connection	133
9.11	Percentage of delivered results for Mobile Edge Computing with the <i>To-Data-First resolution strategy</i> (Baseline).	134
9.12	Percentage of delivered results for Mobile Edge Computing with the <i>Mobile-Edge-Computing resolution strategy</i>	136
9.13	Percentage of delivered results for Mobile Edge Computing with the <i>Mobile-Edge-Computing resolution strategy</i> including Data Upload Strategy for mobile Scenarios.	137
9.14	Summary of the Results of the Simulation of the comparison between Cloud and Edge Computing for Vehicular Scenarios. Time in Seconds.	139
9.15	Summary of the Results of the Comparison the/process function only at the Edge, at a Fog Computing Node and only in the Cloud. Time in Seconds.	141

List of Algorithms

4.1	Sketched Map-Reduce Resolution Strategy.	52
5.1	<i>Resolution Strategy</i> for Mobile Edge Computing.	60
6.1	Choosing the possible subcomputations for the planning procedure.	78
6.2	Choosing the best combination from each possible plan combination. The functions <code>planFWD</code> and <code>planLocal</code> are lookups, since these data have been requested previously.	80
6.3	Creating Clusters of Prefixes to optimize the planning process. . .	82
6.4	Reducing the number of requests for meta information based on clusters.	83
6.5	Analyzing if a plan can be reused.	84

List of Acronyms

AI	Artificial Intelligence
API	Application Programmable Interface
AST	Abstract Syntax Tree
CCN	Content Centric Networking
CDN	Content Delivery Network
CFN	Compute First Networking
CICN	Community Information Centric Network- ing
CS	Content Store
DoS	Denial of Service
FaaS	Function as a Service
FCC	US Federal Communications Commission
FIB	Forwarding Information Base
FLIC	File-like ICN Collection
HTTP	Hypertext Transfer Protocol
IaaS	Infrastructure as a Service
ICN	Information Centric Networking
IM	Interest Message
IoT	Internet of Things
IP	Internet Protocol
IPTV	Internet Protocol Television
ISP	Internet Service Provider
KAM	Keep-Alive Messages
NACK	Non-Acknowledgement
NAT	Network Address Translation
NDN	Named Data Networking

NDO	Named Data Object
NFaaS	Named Function as a Service
NFN	Named Function Networking
PIT	Pending Interest Table
RIB	Routing Information Base
RICE	Remote Method Invocation in ICN
RPC	Remote Procedure Call
RSU	Road Side Unit
RTT	Round-Trip-Time
SCN	Service Centric Networking
SDN	Software Defined Networking
TCP	Transmission Control Protocol
TPU	TensorFlow Processing Unit
V2C	Vehicle-to-Cyclist
V2I	Vehicle-to-Infrastructure
V2N	Vehicle-to-Network
V2P	Vehicle-to-Pedestrian
V2V	Vehicle-to-Vehicle
V2X	Vehicle-to-Everything
VR	Virtual Reality

PART I

Introduction, Background & Related Work

1

Introduction

The Internet is today's worldwide communication, entertainment and information platform. There is no comparable platform in the history of humanity which enabled as many people to communicate with each other while being on the other side of earth. No other technology changed the way how humans are living faster than the digitalization and the Internet. With this, a variety of new industries came up based on digital technology and network communication. No other new technology was expanding as fast as the digitalization all around the globe. While applications of computer networks and the Internet already changed dramatically over the short time they existed, the underlying network architecture was never radically changed.

The architecture has proven to be very robust, scalable and extendable over the time. However, while the architecture itself is scaling, to offer services to a huge number of consumers, additional infrastructure had to be created, since the architecture does not match today's users' requirements anymore (such as video streaming). Therefore, research is ongoing on how to create a computer network and Internet architecture, which better fits today's and tomorrow's users' requirements. Concretely, the focus shifted from connection machines to delivering processed or unprocessed data according to the users' wishes.

This work looks into existing research on how to deliver data from a network more efficiently and proposes ways for efficient *in network data processing* based on this research. Thereby static as well as mobile scenarios are considered. To understand this research, we first need to look at how the Internet evolved and how the offered services changed.

1.1 A History of Computer Networking and the Internet

In 1968 the first computer network, the ARPANET, was designed by the Massachusetts Institute of Technology (MIT) and the United States Department of Defense. It was one of the first network architectures based on a packet-switching methodology and it is the predecessor of today's Internet. When the first computer networks came up in the 1960s and 1970s, the main goal was to connect a single computer to another single computer, to a printer, to another device or to perform a Remote Procedure Call (RPC). Therefore, the network protocols were designed as host-to-host protocols, while packet forwarding was organized in a decentralized manner. For end to end communication ARPANET used the Network Control Program (NPC) to connect two applications on two different machines together.

Internet Protocol (IP) was developed as successor of NPC and was used for host-to-host communication in the Internet. Designed for single host-to-host, single path communication, IP became the standard for basically all Internet communication that occurs nowadays.

While the communication protocols in the Internet stayed the same, the way the Internet was used changed over time. While in the beginning, computer networking was used to connect to a workstation or a printer, later computer networking was used for requesting websites from the Internet. In the early time of the Internet, the World Wide Web was just used to get information and to request passive content. With the advent of social media and Wikis, that changed. Content became personalized and users were adding significant amounts of content to the Web themselves. The main purpose of network connection was no longer to connect to a machine to work or to use a printer, but to connect to a certain machine to fetch data stored on that machine. The service offered by a server became the focus of the users and not the machine itself. Data are stored in highly optimized database systems, which create plans for optimal data access to handle as many database queries as possible at once. With video streaming portals like Youtube and Netflix [AGH⁺15] as well as with Internet Protocol Television (IPTV) the amount of data transported over the Internet increased dramatically. Since the Internet is still connection oriented to download a file, the user needs to address a server storing that file instead of addressing the file directly. Moreover when transferring a large file to many users, the file is transported for each user from the source server the whole way to the client.

This is required since the focus is on the connection, which means the network itself has no information about the transported data. However, for live streams and IPTV the expectation is that many users in the same area request the same data. Thus, a more efficient way would be to transfer data using multicasts.

To handle the huge amount of data generated by video streaming and social media interactions, the Internet infrastructure was extended, but the underlying protocol itself remained untouched. Companies started to build data centers consisting of thousands of servers to handle the amount of requests from a growing number of users. To improve the data distribution Content Delivery Networks (CDNs) were built closer to the users. A CDN is a number of data centers, where each is geographically differently located, so that optimally there is always one close to the users. It mirrors the content of the main data center. By connecting users to the geographically closest data center in the CDN, load balancing is achieved. The data are moved to the CDNs only once and from there they are distributed to the users. Consequently, the number of connections to the main data center is reduced.

While solving the problem of high load on a single connection, CDNs are expensive infrastructure to build. This means, to offer a data intensive service such as video streaming, a large investment in infrastructure is required first.

Summarized, over time the main purpose of network connections changed from connecting machines to transferring files and even data processing and delivering processed files.

1.2 Data Centric Approaches on Network Layer

To better match the users' requirements, alternative network approaches are researched. The idea of these approaches is to improve the network stability, speed and the power consumption by better fitting the requirements of the current usage. The power consumption is an important factor, since today's Internet requires a vast amount energy [HBF⁺11]. For environmental as well as financial reasons a reduction of the energy consumption is required and beneficial.

Most alternative approaches focus on the data delivery and addressing data directly. These approaches are called Information Centric Networking (ICN), since the focus shifted to the delivery of information. The main idea of ICN is to address data directly instead of machines/servers/devices, and the data can be delivered from any devices which stores the data. Research regarding ICN is presented in Section 2.1.

The research in ICN aims to improve the network utilization by integrating multi-casts and efficient data distribution directly into the network. This way the network becomes a CDN for everyone, eliminating the high perquisite requirements to distribute large data objects.

Moreover, there are additional benefits to having data-focused networks. While in today's computer networks and in the Internet, security is implemented on data connections, in a data-focused network security will be implemented directly on the data. This means not only the focus of the network, but also the focus of security is shifted towards the data itself. This has the benefit that the user only trusts the data creator and not the publisher as it is the case in connection oriented networks.

The need for more data oriented systems was already coming up when the World Wide Web appeared. The Hypertext Transfer Protocol (HTTP) is an application layer protocol to transfer data running on top of Transmission Control Protocol (TCP)/IP. It is used to specify which data is requested from a server. This is quite close to the idea of ICN, but ICN offers this functionality on the network layer.

1.3 Computation in Networks

Having huge data centers for content delivery available, providers started selling unused computational capacities. A new network pattern came up – cloud computing. In cloud computing providers offer virtual machines, computers, services, storage or even full private networks in their data centers to be fully managed by the customer. Thus, in the area of cloud computing, the focus shifted from data delivery to data processing. Mobile Clients offload computations to save energy and to accelerate the result production. Some applications – especially artificial neural networks used for image or language processing – require big input data which cannot be stored on a device. This is where Cloud Computing recently also Edge Computing improve the capabilities of mobile devices.

Over time different companies started to offer their cloud services and to improve their offers with features to simplify the usage. Instead of dealing with servers or networks, users can now just define function code and the cloud provider decides where to execute it.

After cloud computing, the Internet Service Providers (ISPs) began to offer computational capabilities, too. This way, the computations move closer to the

clients. This is also called Edge Computing.

Still, the network itself has no computational capabilities and all services run as applications on the existing Internet. Beside the research of ICN, there is research on how to run computation on the network layer itself to improve the stability and performance.

Although the data access in internal databases is highly optimized, access to cloud computing is still based on the simple Internet and Computer Network techniques which are designed to connect single machines together.

1.4 Contributions

This *Thesis* focuses on pushing computations into the network and on its efficient execution. Since usually not the raw data itself but processed data are required by the users, it is more efficient to first process data and then deliver the result. It is very common, that results are smaller than input data.

By giving the network more information about the computation itself and about the required data to compute a result, it is possible to improve the execution of computations by *optimizing the way the computation is executed* according to the requirements of the network.

We start with the existing Named Function Networking (NFN), which is a concept to bring computations into ICN. The idea of NFN is to define a computation and to let the network decide how and where to produce the result. Similar to the idea of ICN, a user just requests a result and it can be produced wherever the network has capacities available. While for data delivery only bandwidth and latency matter, for producing a result many more factors such as computational load on the nodes matter. To produce the results, we take the NFN concept and identify two main components which can be changed independently: Computation definitions and computation distributions. This way computations can be defined independently from the environment it will later be executed in. The main part of this work is defining strategies to execute computations in different scenarios in a way that resources are optimally used. We decided to choose two main scenarios for which we create strategies to execute programs:

- Data Center and Cloud Computing,
- Edge Computing and Mobility.

Thereby, we use two different concepts for cloud computing in NFN. We improve the existing strategy for executing programs, which only considers data already stored on the forwarding nodes. Moreover, we propose a new concept based on creating *Plans* using meta information requested from the network to achieve better execution performance. By looking at load, filesizes, etc plans can be created for every possible situation, such as cloud computing and edge computing. Plans especially have a *potential impact* for cloud computing with long running computations and heavy resource usage. *Plans* are closely related to the query execution planning of databases.

For edge computing, we discovered that the NFN properties to compute a result on any location in a network is a key feature to efficiently handle mobility in e.g. vehicular scenarios. We use NFN as basis for mobile Edge Computing and the underlying ICN for an efficient mobile data upload.

After presenting the technical concepts behind our strategies for efficient execution of computations in NFN, we use our implementation to evaluate and compare these concepts.

2

Background & Related Work

This chapter provides background information about Information Centric Networking (ICN), Content Centric Networking (CCN) and the basics of serverless computing. Moreover, we present the background information to possible use cases such as cloud and edge computing. Thereby, we also provide information to related work and similar projects.

2.1 Information Centric Networking (ICN)

ICN in general is a new communication pattern, which shifts the focus from connecting machines to connecting users to data. The main idea behind this is to match today's communication challenges, where users are usually "interested" in receiving data from the network rather than connecting to a specific machine. Nowadays, to fetch data or to watch a video stream, a user connects to a server storing the data. Next, the user sends a request (usually HTTP) to tell the server, which data they want to receive. There are different flavors of networks focusing on the delivery of data, which all have in common, that a user can directly address the data [ADI⁺12][GSK⁺11], such as DONA [KCC⁺07], PSIRP/PURSUIT [LVT10] or NetInf [DKO⁺13]. Addressing data instead of machines grows the forwarding tables, since usually, multiple data objects are stored on a single machine.

Jacobson et al. [JST⁺09] proposed CCN as an ICN by using hierarchical names to reduce the size of the forwarding table. There are different implementations of the Jacobson et al style ICNs, for instance Named Data Networking (NDN) [ZAB⁺14] and Community Information Centric Networking (CICN). In the following we will refer to these networks as CCN.

In general, CCN uses two different types of messages:

- Interest Message (IM) / Request Message
- Named Data Object (NDO) / Reply Message.

An IM contains the name of a NDO, while a NDO is the combination of a name and a data blob. The binding between a name and the data is fixed, and cannot be changed. Every data blob has its own unique name.

In a typical CCN workflow a user sends an IM into the network to request data with a specific name. The network will search for a NDO matching the name expressed in the IM. If a matching NDO is found, the network will return it on the same path, the IM arrived. Therefore, the CCN communication pattern is pull based, which reduces the risk of “Denial of Service” attacks [GTU⁺13]. The content can be provided by the data publisher as well as any forwarder which temporarily stored the content. This “caching” ability is possible, since the data are bound to their name not to a host. Thus, CCN can distribute popular content objects more efficiently.

A CCN network consists of nodes, which can be either endpoints (e.g. clients, repositories, servers) or forwarders.

In the following we will take a closer look at how a CCN forwarder is designed (Section 2.1.1) and how the CCN forwarding process is implemented in detail (Section 2.1.2).

2.1.1 The CCN Forwarder

A forwarder is the central entity of CCN and responsible for the transportation of requests into the network and reply messages back to the client. Figure 2.1 shows a scheme of the architecture of a CCN forwarder. Usually, it consists of three important data structures:

- Content Store (CS)
- Forwarding Information Base (FIB)
- Pending Interest Table (PIT).

The CS is used to cache recent and popular NDOs to improve their distribution. An entry of the CS is a tuple mapping a name to the data object:

$$\langle \text{name} \rangle \rightarrow \langle \text{data} \rangle \quad (2.1)$$

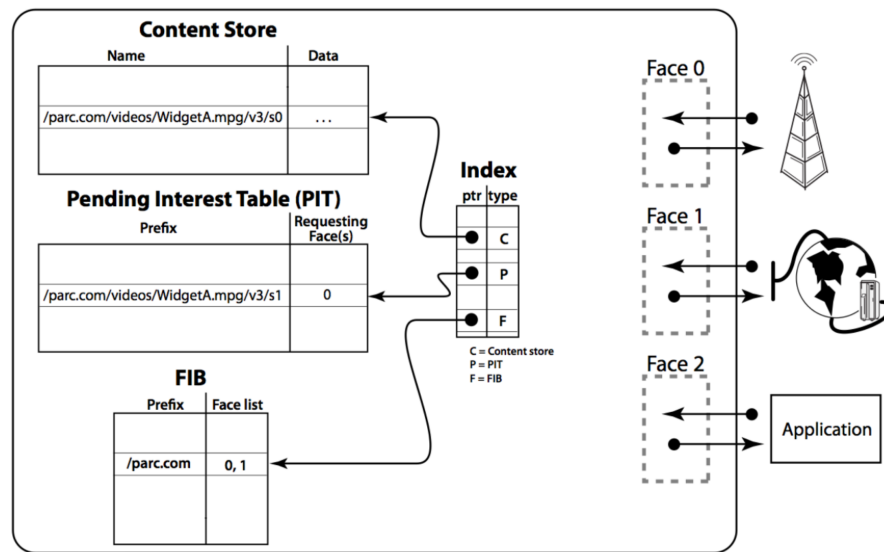


Figure 2.1 Scheme of a CCN Forwarder [JST⁺09].

The CCN Forwarder consists of three important data structures: the CS, PIT and the FIB. It uses virtual interfaces (called Faces) to communicate with other forwarders.

Furthermore, some meta data are stored in the CS, most important the last time the content was used, which is important to decide, when a cached NDO should be deleted. The FIB contains prefixes, which can be matched against IMs to make a forwarding decision. An entry is usually a tuple mapping names to outgoing faces:

$$\langle \text{name} \rangle \rightarrow \langle \text{face} \rangle \quad (2.2)$$

Usually, in CCN longest prefix matching is used for the forwarding process. Thereby, the names in the IM and in the FIB are compared component by component. The FIB entry with the highest number of matching components is chosen. The PIT stores all IM which were forwarded by the forwarder and additionally the face, on which each IM was received. A PIT entry is a tuple mapping a name of an IM to a list of incoming faces:

$$\langle \text{name} \rangle \rightarrow \text{list}(\langle \text{face} \rangle) \quad (2.3)$$

Furthermore, the PIT stores meta data about the time when the PIT entry was created and when an incoming face was appended last to the list of entries. This information is used to decide how long a PIT entry is valid. A face is a virtual interface used by the CCN forwarder to connect to other CCN forwarders.

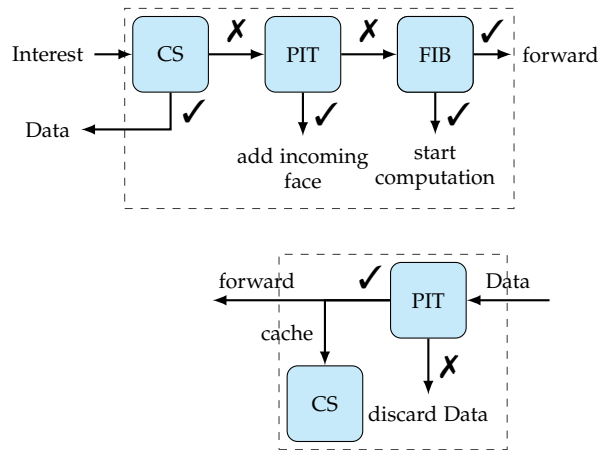


Figure 2.2 The forwarding process of CCN [ZAB⁺14].

If a node receives an IM it first checks if there is a matching NDO in the CS and the request can be replied directly. If no matching NDO is available, the node checks if there is a PIT entry already available, and in this case it appends the incoming face to that entry. If no PIT entry is available the network will use the FIB information to forward the request. If a node receives a NDO it checks if there is a matching PIT entry available, if there is none, the NDO is dropped. If there is a PIT entry, the NDO will be forwarded regarding the entry and it will be stored for a certain time in the cache (Figure by Hamburg University of Applied Science).

2.1.2 CCN Forwarding

The CCN forwarding process is the central algorithm of CCN. The entire process is pull based, which means, that data can only be forwarded if a request for these data is available. It handles two different cases:

- Forwarding an IM to a node which can satisfy the IM.
- Forwarding a NDO as reply to an IM back to the client.

The entire forwarding process of a CCN forwarder is visualized in Figure 2.2.

IM handling: The CCN forwarding process starts when an IM is received. First, the forwarder checks whether there is a matching NDO available in the CS. Thereby, the hierarchical names can be either compared using longest prefix matching or exact matching – depending on the concrete implementation.

If a matching content object is found in the CS, it will be returned to the previous node. Otherwise, a new entry in the FIB will be created. A FIB entry contains the name of the IM and a list of incoming faces. If there is a PIT entry already available for an incoming IM, the face is appended to the entry. In this case, the IM is not forwarded anymore. This enables the forwarder to perform

efficient multi-casts. If a new entry was added to the PIT, the name of the IM is matched against the FIB using longest prefix matching. The IM is forwarded to the best matching entry.

NDO handling: If a NDO is received by a forwarder, it checks, if there is a matching entry in the PIT. If no matching entry is found, the NDO is dropped. Thus, it is not possible to push a NDO without a request, which increases the resistance against “Denial of Service” attacks. In case, there is a matching PIT entry available, the NDO is forwarded using all faces, which are stored in the PIT entry.

2.1.3 Aging of the CCN Data Structures

In the CS and the PIT a lot of additional status – which requires data storage – is maintained in CCN during the forwarding process. In contrast to the FIB, whose status is generated by the network itself i.e. a routing algorithm, the status in the CS and in the PIT is generated by user requests and replies.

To prevent the network from overloading by the status it is required to *age* the data structures.

NDOs are only stored in the CS for a certain time. Popular NDOs can be cached for a longer time than other NDOs. Therefore, a caching strategy is used. A simple caching strategy is to increase the caching time for every entry which is hit by an IM and could be reused.

The status maintained by a PIT is comparable with the status maintained by a Network Address Translation (NAT), where the mapping between the previous and the new address is stored for the time the connection is valid. To protect the PIT from overloading, the entries in the PIT are only valid for a short time period. In this time period the request is forwarded several times, to minimize the risk of a *packet loss* due to network errors. Usually a PIT entry is valid for a bit more than the expected Round-Trip-Time (RTT). If another IM with the same name is received, the PIT timer is reset. Thus, a PIT entry will be valid, as long as a client tries to poll for a result.

If no reply message to an IM is received, the PIT entry times out and is removed. If the CCN implementation supports Non-Acknowledgement (NACK) messages, a NACK message is used to notify the previous nodes, that the request cannot be satisfied. NACKs are also used if there is no forwarding rule for a message or content is not available.

In general, both CS and PIT have a maximum number of entries which can be stored. In case this number is exceeded, requests will be rejected or old entries will be deleted, even if there was not yet a timeout.

2.1.4 Chunking

In CCN, NDOs are usually not delivered as a single packet, but they are split in equal sized chunks, which are transported over the network. For example, chunking is important for the caching properties of CCN, since it could be complicated to cache an entire NDO if it is relatively large. By using chunks, larger NDOs are split and this way parts can be cached. Therefore, the *large file* problematic is resolved.

Furthermore, by chunking NDOs the packets sent via the CCN have a smaller size, which means, it is less overhead to retransmit a single packet, because of a network error or a flipped bit.

In different CCN implementations different ways of chunking have been proposed. In general a chunking implementation is independent from the implementation of the CCN, but rather depends on the concrete application and their requirements.

A simple chunk implementation can be splitting a NDO into n chunks so that $\text{size}(\text{chunk}_i) = x, i = 0..n - 1$ and $\text{size}(\text{chunk}_n) \leq x$. Each chunk can be identified by adding a name component containing i as *chunk number* to the end of the name. A special marker will be added to the last chunk, so that a client can understand that it received all chunks. This simple approach creates little overhead, but it works best, when chunks are requested in order, for example when streaming data.

Other approaches offer more flexibility to randomly access any chunk. For example, manifests are used to store an index structure. A manifest is just a simple NDO storing references to other NDOs containing either data or further index structures. A scheme of such an index structure is shown in Figure 2.3. Usually, a root manifest is available under the name $\langle \text{name} \rangle$ without extensions. The root manifest contains pointers to the chunks, which are often identified by $\langle \text{name} \rangle / \langle \text{hashvalue} \rangle$. Furthermore, the root manifest and every further manifest can contain pointers to other manifests, so that a tree structure is created. Each manifest can be identified by a name $\langle \text{name} \rangle / \langle \text{manifest-name} \rangle$.

Beside the pointers to chunks or other manifests, a root manifest can also contain meta information about the corresponding data such as total size or total number of chunks.

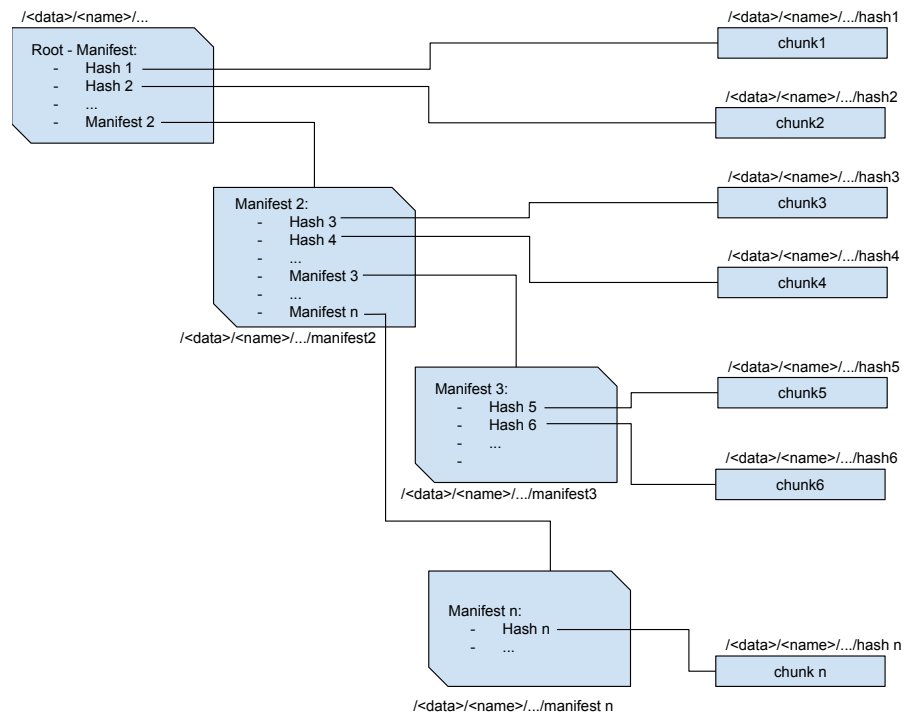


Figure 2.3 Scheme of an index structure for addressing chunks as it is used by File-like ICN Collection (FLIC) [TS14].

An index structure can be used to enable clients to access chunks easily and fast, even when accessing them out of order. In this case, the chunks are identified by their hash value. A client learns about the hash values by first requesting the manifest, which is a NDO with the data name with no additional name components. It contains hash values of the chunks and it can refer to the names of further manifests. Therefore, random access becomes more efficient, compared to a ordered list of numbered chunks.

File-like ICN Collection (FLIC) [TW16] is an example for an index structure, which enables users to access chunks similar to accessing files on a file system.

2.2 Named Function Networking (NFN)

CCN describes a way to request data from the network independent of the location where the data are stored (see Section 2.1) by requesting the data directly using their name [TS14]. Generally spoken, requesting static, unchanged data is only a special case of requesting dynamic processed data. In the time of Cloud computing, data are often stored within data centers and not on local machines. If data are requested by a client, these data are often downloaded and transformed afterwards. Since Cloud data centers come up with high computational power, computing results within the data center instead of computing on local

machines can be useful, especially since this is often more efficient since the size of processed data is usually smaller than the input data size (since something is computed out of the data and no new data are generated). For example, Apache Hadoop uses the pattern to execute computation in a data center close to the input data [IKAM17]. Furthermore, the result can be reused for similar requests. For example, to save bandwidth and to reduce the load on the network the resolution of a video can be reduced before it is delivered to a mobile device.

NFN describes a generalization of CCN, where users do not only define which data they want to request, but also how the data should be transformed. Similar to CCN, where a data name is encoded in a request and the network finds a way to deliver the data, in NFN a computation is defined within the request, and the network finds a way to “cook” the result. In CCN a request is simply forwarded according to the FIB. In NFN the forwarding process becomes more complex since it includes the execution of computations. The forwarding of an IM determines where a computation is executed. Thus, we call the mechanism of deciding how and where to compute a result *resolution strategy*.

In general, NFN consists of two independent key components:

- A Query Language which defines how to express a computation and how to encode a computation in the name of an IM.
- A *Resolution Strategy* to decide how a NFN request is forwarded and where the computation encoded in the request is executed.

The following two Sections will address these components.

2.2.1 Encoding Computations in CCN names using the λ calculus

To enable the network to execute a computation on any node with computational abilities, the required *program code* to be executed has to be transported to that node. In NFN, the program code is stored in so called *Named Functions*. A Named Function is a named reference to executable code, for example a NDO which contains the program code. To call a Named Function an IM is issued containing the name of the function and the names of the function parameter which are regular NDOs or further function calls. Furthermore, primitive data types, such as integer, floats, strings, etc can be used as parameters, too. With function and parameter names a NFN request can contain more than one name. Unfortunately, in the underlying CCN the FIB only supports forwarding of IMs

which contain exactly one name. Therefore, it is required to transform the name of a NFN request, to restore the compatibility with CCN forwarding [SKS⁺14].

2.2.1.1 λ Calculus Basis

To encode a NFN computation in the name of an IM, the λ calculus can be used. The λ calculus is a formal way to encode mathematical and logical expressions [Wik18]. It has been proven that the λ calculus can be used to encode *Turing complete* computations [Tur37].

The λ calculus is defined by three basic operations:

- a *variable* x ,
- an *abstraction* $\lambda x.M$, where x is a variable and M is another expression in the λ calculus,
- an *application* $M N$, where M and N are both λ calculus.

A grammar for the λ calculus is:

$$M = x \mid \lambda x.M \mid M N. \quad (2.4)$$

For the purpose of NFN, the λ calculus is extended to support CCN names and with a possibility to call *Named Function* i.e. a CCN name of a NDO containing program code. Thus, for NFN the λ calculus is defined as:

- a *variable* x ,
- a *CCN name* $\langle \text{name} \rangle$,
- an *abstraction* $\lambda x.M$, where x is a variable and M is another expression in the λ calculus,
- an *application* $M N$, where M and N are both λ calculus,
- a *function call* $\text{call name } M^*$, where name is a CCN name and M^* is an arbitrary number expressions in the λ calculus which are used as parameter for the function call.

The grammar for the extended λ calculus for NFN is:

$$\begin{aligned} E &= \langle \text{name} \rangle \mid C \\ C &= \text{call } \langle \text{name} \rangle P \\ P &= E * \mid x \mid \lambda x.M \mid M N \end{aligned} \quad (2.5)$$

2.2.1.2 Transforming Expressions for NFN

A NFN expression can be either a simple CCN *name* $\langle \text{name} \rangle$ or a *Function Call* C . A *Function Call* consists of the name of a *Named Function* and further expressions, which are the parameters.

This grammar requires at least one *Function Call* or at least one CCN *name*, since a pure λ calculus is not meaningful for a CCN network.

Thus, a function call in NFN has the form:

`call <functionname> <p1> ... <pn>`,

where n is the number of parameters.

To overcome the limitation of the FIB, which can only handle IMs containing a single name, we take advantage of the *longest prefix matching* used by the forwarding process of CCN and of the abstraction, as well as of the application of the λ calculus [SKS⁺14]. Applying an abstraction to a name of the expression means a variable is introduced for the name:

`call <functionname> <p1> ... n ... <pn>`

is transformed to:

`(λ x. call <functionname> <p1> ... x ... <pn>) n`,

where n is a CCN name. When the name is applied to the new variable, the original expression is restored. Unfortunately, the expression is still not meaningful to CCN forwarding. But we have a name n isolated at the end of the expression. If this name n is moved from the end of the expression to the front, the expression is transformed in a way, that it starts with a CCN forwardable name:

`n (λ x. call <functionname> <p1> ... x ... <pn>)`.

To encode this expression in the name of an IM, the name n is put into the first components, where each component of n is encoded in a name component. Furthermore, there is an additional component added to the end, which contains the letters NFN (NFN tag) and is used to identify an IM as a NFN request. Thus, the name which is encoded in an IM is:

`n | (λ x. call <functionname> <p1> ... x ... <pn>) | NFN`,

where `|` marks the split between two components. Since the CCN forwarding process is based on longest prefix matching, the FIB entries are matched against

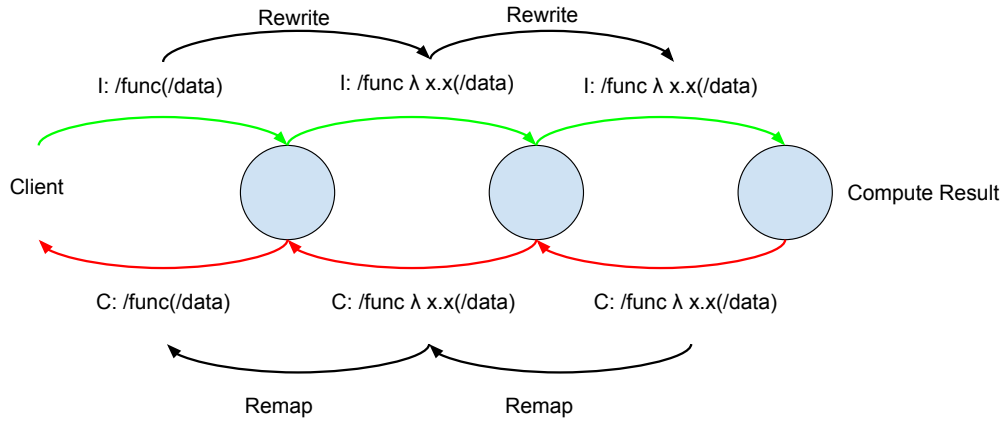


Figure 2.4 Rewriting and Remapping process in NFN.

the prepended name in the first components, while the rest of the expression and the NFN marker are ignored. For example, the computation

```
call /lib/func/f1 /data/d1 /data/d2
```

can be transformed to

```
data | d1 | (λ x. call /lib/func/f1 x /data/d2) | NFN.
```

NDOs containing the results are stored under the name used to encode the computation in the IM. If the result is transported on the trace to the requesting client all name transformations – the network may have applied – are remapped, so that the client gets the NDO with the same name as the original IM had. A scheme of the rewriting and remapping process is shown in Figure 2.4.

2.2.2 Resolution Strategies and Pinned Functions in NFN

After defining how to encode a computation in an IM, and how to transform an expression in a way, that it is meaningful to CCN forwarding, it is required to define which name is prepended in front of the expression. This is important since CCN forwarding supports only single names while NFN expressions consist of multiple names.

In NFN neither the user nor the client application is responsible for prepending a name. The NFN nodes in the network prepend a name or even exchange the prepended name to determine where a computation is executed. Figure 2.5 shows two cases of executing a computations in NFN:

1. find cached or produce a result at the data location, fetch the function code (1),

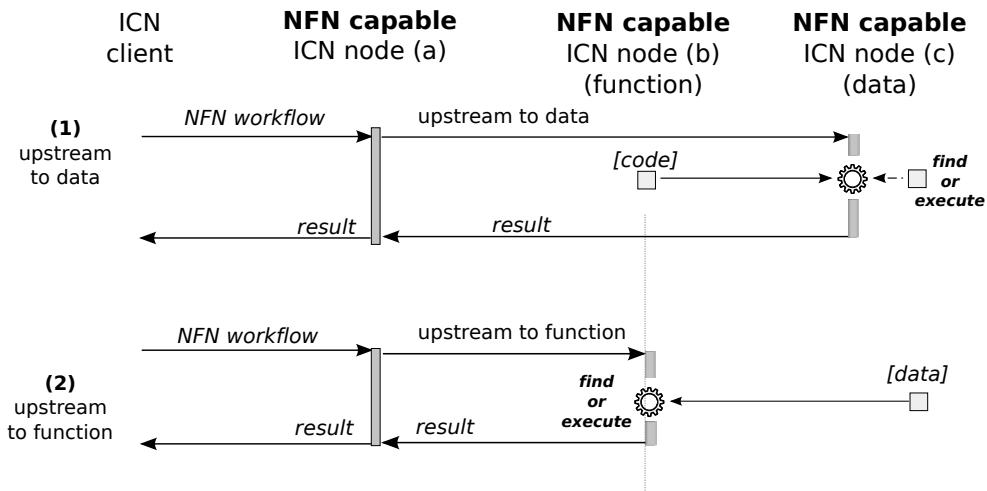


Figure 2.5 Two cases of NFN execution.

2. find cached or produce a result at the function code location, fetch the input data (2).

When it comes to data processing, usually, the size of the input data is large, while the size of the function code is more compact. Since the function code is encoded in NDOs it is not required to transport the data to an execution node which stores the function code, but the function code can be transported to the nodes storing the data, under the assumption that the result of a computation is smaller than the size of the input data.

For NFN it is easy to forward a request to a node that stores the data. It is only required to prepend the input data name and the underlying CCN will transport the request to a node that stores the data. For requests with multiple CCN names as parameters each parameter is valid. Thus, for the basic NFN, the first parameter, which is a CCN name, is chosen [SKS⁺14] ((1) in Figure 2.6). Each node in the network will forward the interest message according to the FIB entries ((3) in Figure 2.6). If a node has the NDO matching the prepended name in the IM locally stored, it will start the computation ((2) in Figure 2.6). Therefore, the node sends out IMs for the other parameters and the function code ((4) in Figure 2.6). As soon as all required NDOs are available, the node will execute the computation ((5) in Figure 2.6). In case, that a parameter is another function call, an IM containing the subcomputation is sent into the network and executed on another node. The result of a subcomputation is stored in a NDO to be transported back through the network ((6) in Figure 2.6).

However, Named Functions only support side effect free computations, which have no dependencies to the executing machine, so that they can be executed on

any node. In case, the function requires a private key or other secret information – that cannot be exposed to the network – it has to be pinned to a node. Thus, it is not possible to execute such a pinned function on the node where the input data are stored. Therefore, the data have to be transported to a node where the function is pinned to. To do so, the name of the function has to be prepended instead of a data name. Since a node does not know, if a function is pinned or not and this should be transparent to the user issuing a NFN request, the network needs a strategy to decide, if a data name or the function name should be prepended.

Since computing close to the location where the input data are stored is more efficient in regard to the amount of data transfers, the network first tries to forward an IM by prepending a data name. Only if the network cannot compute the result on the location where the input data are stored, the network will try to compute the result on the location where the function is stored (or pinned) by prepending the function name. The network learns, that a result cannot be computed at the data location by a timeout or by a NACK message ((7) in Figure 2.6). Beside their usage in CCN, in NFN NACK messages are used to advise previous nodes, that a computation cannot be executed. This could indicate a *Pinned Function*, which requires the request to be forwarded to a node on which the function is pinned to((8) in Figure 2.6). In this special case, in which the result cannot be computed on a node storing the input data and on a node that stores the function code, a timeout or NACK will notify the client, that the result cannot be delivered ((9) in Figure 2.6).

2.2.3 Timeout Prevention in NFN

When serving static data using CCN the PIT timeout is defined by the RTT. For NFN this is not sufficient, since the overall runtime of a computation is unknown. Unfortunately, if there is a timeout of a PIT entry, the trace on which a NDO would be delivered is gone. Therefore, there is no way to ship the result, if there was a timeout by just relying on the existing mechanism. In NFN, there are two major approaches for timeout prevention:

- Thunks can be used to check, if a result can be computed. A thunk is returned to the requesting client with a time estimation.
- Keep-Alive Messages (KAM) can be used to find out, if a computation is still running. In case, it is still running, the original request can be repeated to renew the PIT entries.

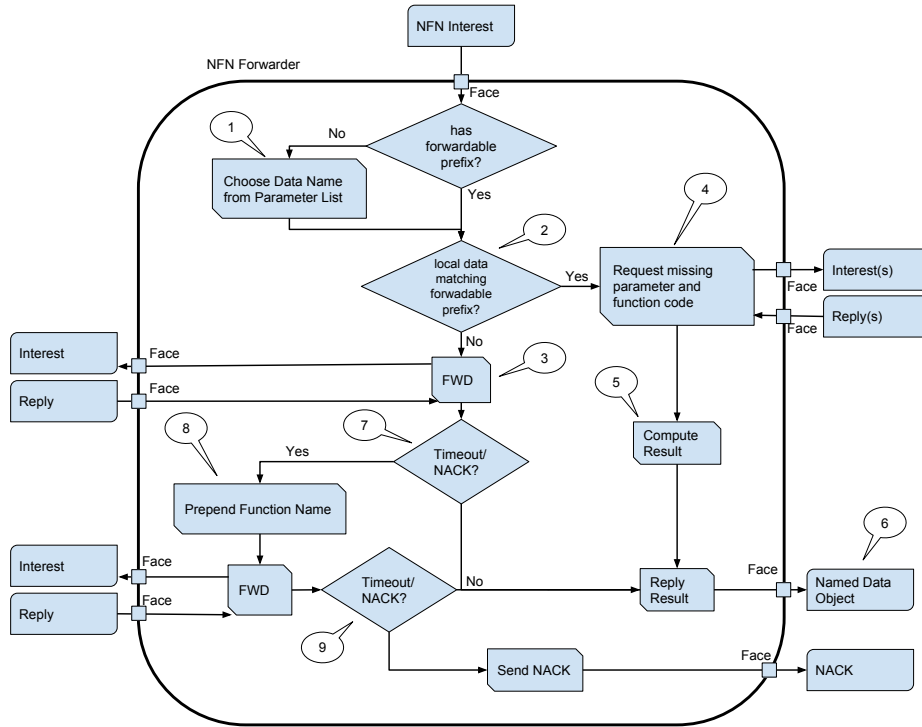


Figure 2.6 Scheme of a NFN resolution process.

When a forwarder receives an IM it checks if there is a forwardable prefix. If there is none, the node will choose one and prepend it (1). Next, the node will check if the prepended name is locally available (2). If it is the node will issue requests for the required parameters (4) and compute the result (5) and reply it (6). If the data are not available, the node will forward the request (3). If a result is found, it will be replied (6). Otherwise, if there is a timeout or a NACK (7), the node will prepend the function name (8), for the case that the computation failed, because the function code is pinned. If now a result is found it will be replied (6). Otherwise the computation has failed and there will be a timeout or a NACK is replied (9).

There might be additional valid approaches for timeout prevention. The only requirement is to provide a way to transport the result. In the following two sections, the Thunks and the KAM approaches are presented in detail.

2.2.3.1 Thunks

A Thunk is a promise given by the network, that a computation can be executed. It is derived from the term “has thought about” [SKS⁺14]. If a Thunk is requested the network checks, if it is possible to satisfy all requirements to compute the result. In case all requirements are satisfied, the network will reply a Thunk to the client containing a new name n_1 (a link used to request the result) and an estimation of the required time to compute the result ((1) in Figure 2.7).

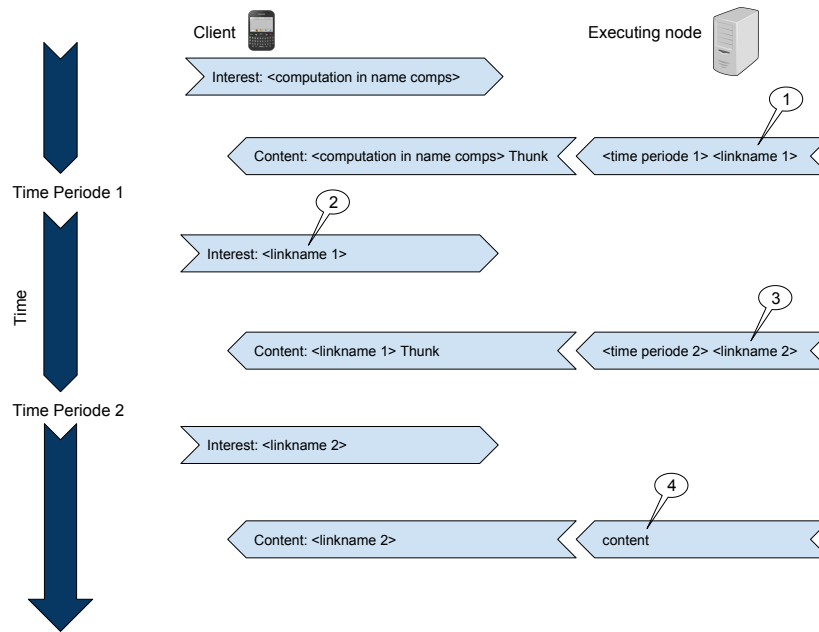


Figure 2.7 Resolving links when receiving Thunks instead of the final result.

After sending the IM the network replies with a Thunk (1) containing a time period and a link. The client waits the time period and sends a new IM with the name of the link (2). In case the result of the computation is not ready, the network returns another Thunk (3). In the other case when the result is ready, the network replies the result (4).

After the time defined by the Thunk, the client will issue a request for the new name n_1 ((2) in Figure 2.7). If the result is ready by now, the network will reply with the result ((4) in Figure 2.7), otherwise it will reply with another Thunk with a further link and another time period [KHO⁺18] ((3) in Figure 2.7).

The huge advantage of Thunks is, that the receiver driven communication is preserved, and only when there is actual communication, a status is generated in the network (There are only PIT entries created when the client requests a Thunk or the result). On the other hand, Thunks introduce new names for results, which can result in a situation, where the result has a different name (given by the link in the Thunk) than the initial request. Therefore, when requesting a result from cache, a client has to follow the chain of all links to receive a cached result. Since the client does not know, that the Thunks were delivered from the cache of the network the client probably waits the time period given by the Thunk instead of directly following the link. Compared to the client requesting the result for the first time which required the Thunks for timeout prevention, for all clients, using the cached result, resolving these links are just overhead. Figure 2.7 visualizes how to deal with the links received within a Thunk.

2.2.3.2 Keep Alive Messages (KAM)

KAMs are used to steer computations and to keep the PIT entries alive [SFT17]. The goal of KAMs is to check if a computation is still running and to inform the executing node that the computation is still awaited by a client. A KAM has the same name as the original request, but carries a marker expressing that it is a KAM. The message is transported to the node executing the computation ((1) in Figure 2.8). In case the computation is not running anymore, a NACK message will be replied or a timeout occurs. If the computation is still running, an empty NDO is applied carrying a marker that it must not be cached ((2) in Figure 2.8). If a client receives the reply to a KAM, it expresses the original IM again, to renew the PIT entries ((3) in Figure 2.8). The client periodically has to send the KAM using a smaller time interval than the PIT for its timeouts. This way, the PIT trace from the executing node back to the client is preserved. In case, the executing node does not receive a KAM for a certain time, it can abort the computation, since the client seems to have disappeared. However, there are scenarios, where it is useful to compute the result anyway, for example in environments with very unreliable connections or for delay tolerance. The big advantage of KAM is that the status of the client is passed to the executing node. However, there is additional status in the network, since the PIT entries exist for the entire runtime of the computation. A PIT entry contains at maximum the number of interfaces the forwarder has, thus the entry size is constant to the number of clients requesting the same result and grows linear with the number of different computations.

2.3 Computing in ICN Networks

NFN is designed to bring computations into the world of CCN. Beside NFN, there are implementations trying to bring computing to CCNs, such as Service Centric Networking (SCN) [BHH⁺11], Named Function as a Service (NFaaS) [KP17], Remote Method Invocation in ICN (RICE) [KHO⁺18] or Compute First Networking (CFN) [KMO⁺19]. Moreover, we take a look at a concept about caching and optimizing computations developed by Intel. Serverless computing in CCN takes advantage of the fact, that CCN does not rely on host names, so that the execution location and the program definition are independent.

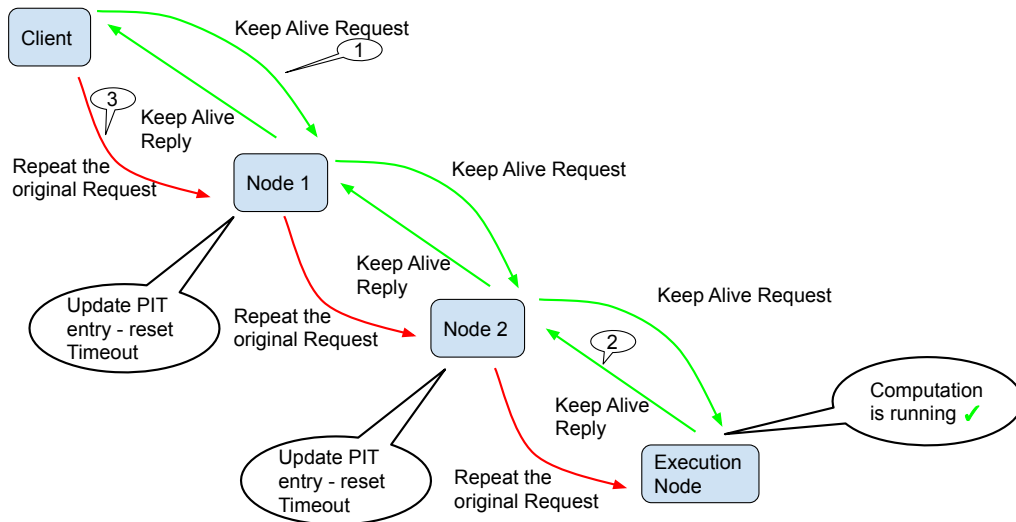


Figure 2.8 Keep Alive Messages to prevent PIT timeouts for computations.

The client periodically sends a KAM (1). In case the computation is still running the network replies with a KAM reply (2). As soon as the client receives the KAM reply it retransmits the original IM (3), which will reset the PIT timeout counter, so that the PIT entries are not deleted.

2.3.1 Service Centric Networking

SCN [BHH⁺11] is the first approach to network computing in CCN. The idea of SCN is to add nodes with computational abilities into the network and to offer services on them. To invoke a service an interest message is sent starting with the name of the service to be invoked. The names of the input data are encoded inside the name components following the name of the service:

```
/name/of/service/name/of/param1/name/of/param2
```

Therefore, SCN does not support subcomputations.

When a request reaches a node which offers the service, the node will request the parameter and invoke the service. Thereby, the data is always transferred to the service, since SCN does not support mobile code.

2.3.2 Named Functions as a Service

NFaaS [KP17] is a computation system for ICN which uses microkernels for the function code. Each microkernel is named and can be requested over the network. The workflow definition is similar to NFN. NFaaS was tested as framework for offloading Virtual Reality and Augmented Reality. Thereby, the first node receiving a request will handle the computation, except it has too much load, then it will offload the computation further to a neighbored node. Micro-

kernel as execution environment are very safe since they are isolated by virtualization and they can very quickly boot.

2.3.3 Remote Method Invocation in ICN

RICE [KHO⁺18] is basically an extension to NFaaS for remote procedure calls. The idea of RICE is to call a microkernel close to the user (edge computing), where the user provides the input data. This happens in a setting, where the user cannot be addressed by a name. To upload data, RICE uses long lasting PIT entries which are bidirectional. To invoke a function call, RICE uses a 4-way handshake. Thereby the user first request the computation, which is replied by a Thunk (a promise the result will be available). The Thunk contains the name of the result. After sending the Thunk, the server will request the input data from the user by using the bidirectional PIT entries. RICE is very flexible since it does not require a name for the user to upload the data, however the bidirectional PIT entries may be a problem if the client is mobile and changes its location fast.

2.3.4 Compute First Networking

CFN [KMO⁺19] is a framework for cloud computing in ICN. It uses microkernels to execute functions. The novelty is a task scheduler, which distributes computations over the network. The task scheduler tries to execute computations close to the input data to reduce data transfers and the network traffic. Moreover, CFN adds additional features, such as state to the functions, which can be maintained over multiple function calls.

CFN perfectly fits into the concept of NFN that separates the workflow definition from the actual execution engine. Thus, the task scheduler is basically a resolution strategy in the concept of NFN.

However, due to the functional manner of the NFN workflow definition, NFN does not support any maintainable status within functions.

2.3.5 Intel: Joint Optimization of Routing and Caching in Heterogeneous Wireless Networks

In this concept, for performing computations, there is a stable area defined in the network [GY19]. Within this stable area, data is instantly available. This approach tries to reduce the load on the network by being throughput optimal. From this point on, adaptive algorithms are used to speed up computations.

2.4 Database Query Optimization

Database queries are usually based on a standardized language to describe the database requests. Usually, database queries are described with the domain specific language SQL. A SQL query contains instructions and database/table identifiers. With this information, the database management system can identify the data requested by the user. SQL offers basic operators for requesting for inserting, editing and for deleting data. Thereby, the data can be distributed over multiple tables. When requesting data, they are collected from all tables. To request data efficiently, it is necessary to optimize the queries in a way that the database can efficiently execute it. This is done by a database query optimizer [Cha98]. Beside SQL which is only used in relational database in other databases, similar mechanism are used.

2.5 Serverless Computing

Serverless Computing or Function as a Service (FaaS) is a current industrial trend, pushed by leading cloud providers. It is a further abstraction to the cloud solutions so far: Infrastructure as a Service (IaaS) describes virtual machines, data storage as well as (sub)networks in the cloud while container (e.g. Docker) based solutions are a first abstraction, focused to reduce the effort to launch additional instances, but gives full control over the system in the container. In contrast, FaaS describes services, where a single function is defined to be executed on the computation infrastructure of the provider [BCC⁺17]. The management of the servers, operation system and other resources are handled by the provider transparent to the application programmer [Ari18], while the application programmers can focus entirely on the development of application logic. Thus, FaaS brings advantages for the cloud provider and for the application programmer. The cloud provider can dynamically choose an execution location according to the current load in their computation infrastructure. The application programmers are released from the management of the infrastructure to execute a function and the function can automatically be scaled to an arbitrary number of nodes. This is possible by simply calling multiple instances of the function on different nodes.

This property is feasible, since FaaS follows the pattern of functional programming. Thereby, a function requires some properties to be executed in a serverless environment [Ari18]:

- Hard limit for the maximal execution time (currently it is common practice among all cloud providers to have a time limit). In theory there is no need for a hard time limit, only the requirement that the function must terminate, a time limit is one way to circumvent the “halting problem” [Dek⁺59]).
- Stateless execution, so that the function can be called on any node.

A function is defined by the function arguments and how these are transformed. The context of a function describes the required runtime environment, timeouts, memory limits, etc. The output of one function can be used as input for another function, where both functions can be developed in different programming languages [Ser18]. The functions are not predefined by the cloud provider but are developed by application programmers, which want to use the FaaS.

2.5.1 Existing Research in Serverless Computing

To create a function, cloud providers usually offer a web based User Interface, where customers can define their own functions. Often, such User Interface offer a method to create a function with a minimum amount of work, for example enabling application programmers to easily import data from other cloud services. Furthermore, an Application Programmable Interface (API) is offered which defines how to invoke an existing function. This is usually done by triggers. For example, a client can invoke a function using a HTTP trigger (meaning the client sends a specific HTTP request to a server of the FaaS provider). Amazon offers a REST based API, where a caller can define the input parameters which define how the result is delivered [Ama18]. Typically, the input data are either delivered directly by the function caller or are received from the Internet (e.g. from a database). In the case of Amazon, it is possible to interact with other cloud services offered by Amazon. Thus, alongside explicit calls by a client using the HTTP triggers, a function can be invoked by various other triggers, for example if an entry is added to a database or based on a Cron job.

Today, almost every cloud provider offers a FaaS system (incomplete list):

- Amazon Lambda [Ama18]
- Google Cloud Functions [Goo18]
- IBM Cloud Functions [IBM18]

- Microsoft Azure Functions [Mic18]
- Oracle Cloud Fn [Ora18]

Beside the commercial solutions there are Open Source implementations available, such as Fn Flow [Pro18], which can be used to setup a local FaaS environment.

2.5.2 Use Cases of Serverless Computing

Serverless computing is often used for data transformation. A common use case is the transformation or the filtering of data objects delivered from the cloud. Thereby, the data are stored in the same cloud as the serverless function lives in. Here, the advantage of processing the data before delivering them is that usually by processing the data size is not increased, often the data are filtered and only a small fraction is delivered to the user. This way the load on the network can be reduced. One example for this kind of operation is the transformation of a video to reduce the size and match the requirements of a mobile client [Dzo18]. A further use case is to scan the logs of machines or servers running in the cloud for errors and triggering warnings, if any error occurred or if an update is available [Med18]. The advantage of this use case is, that the user is not required to run and maintain a monitoring server. Furthermore, serverless computing can be used to trigger automatic backups.

A very popular use case of Amazon Lambda is processing of queries issued by their automated home assistant “Alexa”. Thereby, “Alexa” issues a query, which triggers the call of a function. The function may issue other queries, for example it could do a “Google Search” or other database requests. The resulting data can be filtered and processed by the function and returned to “Alexa”. This enables application developers to define own “Alexa Skills” (additional functionality) without maintaining own servers or having any infrastructure, but by only defining the functionality of the “Skill” itself.

2.6 Edge/Fog Computing

Edge Computing (sometimes also called Fog Computing) is a special variant or extension of cloud computing [SCZ⁺16].

In the literature, Edge Computing is sometimes described as computing on the nearest node in the network (e.g. on the base station), while Fog Computing

describes computing closer to the client than cloud computing, but not necessarily directly next to the client. Here we focus mainly on pure Edge Computing, however, the presented principle can be applied to Fog Computing, too.

Cloud Computing describes central data processing or data storage in the core of a network. It is characterized by high computational power, but since it is in the core of the network, the latency between a client and the cloud is quite high.

Edge Computing tries to move data processing as close to the client as possible, often only one or two hops away, often directly on the Gateway. Therefore, Edge Computing offers low latency, but usually the computational power on a single Edge Device (Edge Gateway) is smaller than in the Cloud.

Edge Computing entities can forward computation requests deeper in the network into a Cloud Data Center, if necessary. Since Edge Computing especially focuses on low latency, it is required to find a fast way to decide if a computation is executed or forwarded.

Figure 2.9 shows a scheme of Cloud and Edge Computing.

The trend to low power Internet of Things (IoT) devices is a driving factor for Edge Computing [SGF⁺10]. These devices can use gateways with computational ability to offload compute intensive operations. Usually, many different IoT devices share a few Edge Gateways. This way, the selling costs of the IoT devices can be reduced, while it is only required to have a single device with an expensive and computational powerful processing unit. Furthermore, this property simplifies hardware upgrades e.g. to add TensorFlow Processing Unit (TPU) for Artificial Intelligence (AI) and Machine Learning, since only the Edge Gateway must be upgraded.

Scenarios, Edge Computing is focusing on, are usually characterizable by constraint (IoT) devices, which cannot perform compute intensive operations and by one of this two key properties [SCZ⁺16]:

- The application requires a fast reaction time. For latency reasons, computations cannot be offloaded into the cloud.
- The application at the edge produces a huge amount of data. Thus, it would be *unhandy* to transfer them into the core of the network.

Both of these properties cannot be served by using the existing Cloud Computing Infrastructure, since the location where the data are stored/produced changes compared to classical cloud computing, where the data are stored in the core of the network and – processed or unprocessed – delivered towards a

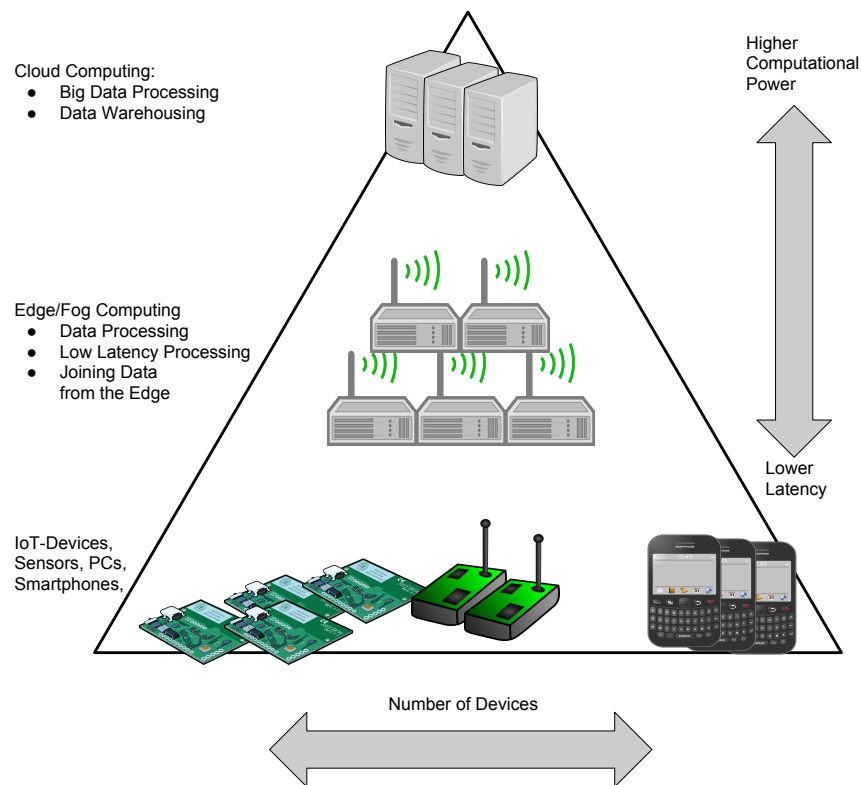


Figure 2.9 Scheme of Cloud Computing and Edge Computing [Lea18].

There are millions of devices with low computational power, there are ten of thousands Processing Units at the Edges, and there are thousands of Processing Units in the cloud. However, the computational power of a single client device is lower than the computational power at the edge and the computational power of a device at the edge is lower than the computational power of a device in the cloud.

client. In IoT scenarios, the data are produced by sensors and other (low power) devices at the edge. The different data flows of Cloud- and Edge Computing are visualized in Figure 2.10.

Since Edge Computing is geographically distributed and associated with the location of the client device, the load is geographically balanced. Furthermore, due to the absence of a central entity, the geographically distribution and the proximity to the client, Edge Computing is more robust against network failures than cloud computing.

Typical practical use cases of Edge Computing are for example [BMZ⁺12]:

- Connected Vehicles: Vehicles collect data about the road conditions and transfer them to the Edge to warn other vehicles about hazardous conditions (See Section 2.7).

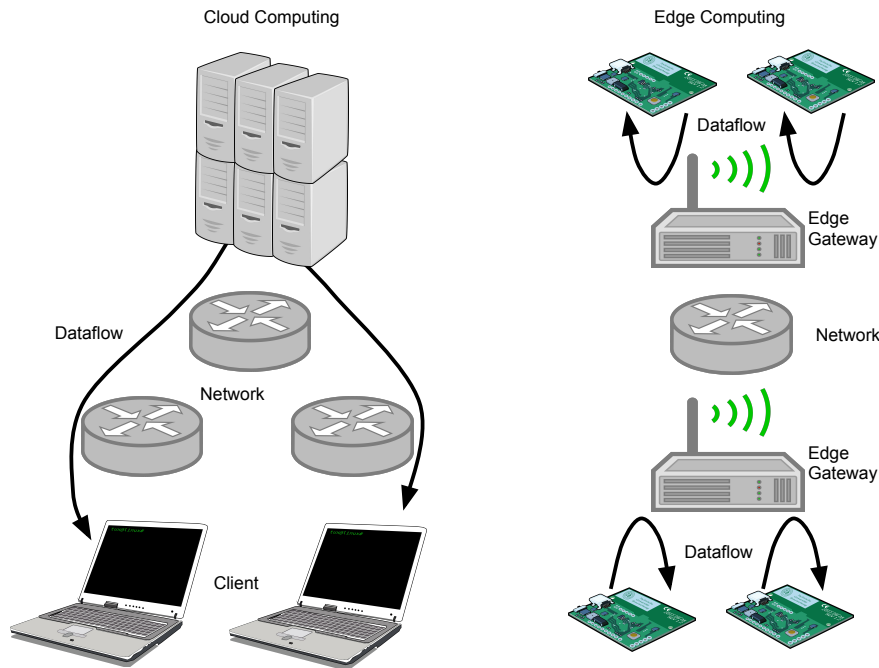


Figure 2.10 Difference in the typical data flows in Cloud and Edge Computing.

The data flow of cloud computing is usually from the Cloud to the clients. In the Edge Computing case the data are produced at the edge and are not transferred into the core of the network.

- Home Automation and Monitoring: Smart Home devices are usually low powered, so it makes sense to offload some operations. The Edge Gateway can be used as *bridge*, so that the Home Automation devices can be controlled from the distance.
- Sensor Data Processing: Large Scale Sensor networks often require data processing or data preprocessing (filtering) before data are transferred into the cloud.
- Virtual Reality (VR) Devices: Today, most VR devices are connected to a computer by wire and only serve as a “monitor”. In future, these devices could be low power computers and offload computational intensive operations using Edge Computing [Bur17][Sat17].
- Industrial Control Systems (ICS) : The industry 4.0 requires automatic production and controlling of production processes. Edge computing can be a driving factor for the automation of industry processes [PIUB⁺17].

Furthermore, computing at the Edge brings new players into the game of network computing. While in cloud computing, mainly Amazon, Google and Microsoft are dominant, edge computing brings chances for ISPs to enter the game.

Moreover, edge computing can reduce the energy consumed by the network, since the number of hops for data transfers can be reduced.

2.7 Vehicle-To-Everything Communication and the Electronic Horizon

Reducing the number of traffic related fatalities is one of the primary goals of integrating driving assistant systems into vehicles. Nowadays, modern vehicles are equipped with various sensors such as lane assistants or emergency breaking systems.

The ultimate goal is the development of autonomous vehicles, which do not require any human interaction. A “driver” just enters the destination, where the vehicle should drive to, and the vehicle will do so. This would also enable Taxi companies to offer Transport as a Service with autonomous vehicles, such as Uber and Lift already offer today with human drivers.

Thus, in future, the degree of automation will increase, and so does the number of sensors to collect data about the road conditions and the traffic ahead. Such data cannot be only used by the vehicle which collected the data but by adding these data to real time maps, they can be shared among all vehicles on a road section to increase the safety for everyone [Ger12][WSG⁺14].

Since dangerous situations can occur unpredictably, it is required, that the data about the current traffic situation are shared with as a small delay as possible and the transmission should be as reliable as possible. Furthermore, since autonomous cars produce a huge amount of data, it is difficult to transport the data into the cloud [Nel16]. Edge Computing, i.e. Mobile Edge Computing is a way to perform computations on the data shared by the vehicle. Therefore, it would be required to deploy edge computation entities along the road.

2.7.1 Road Side Units (RSUs)

In vehicular scenarios computing entities along the road are called *RSUs* [LSW⁺08]. More specific, a kind of RSU is a cellular base station or WiFi access point with computation ability.

Usually, RSUs are connected to each other and can share data. This is important, since due to the high velocity of a vehicle the connection time to a single RSU is often too short to transfer data, start a computation and receive a result.

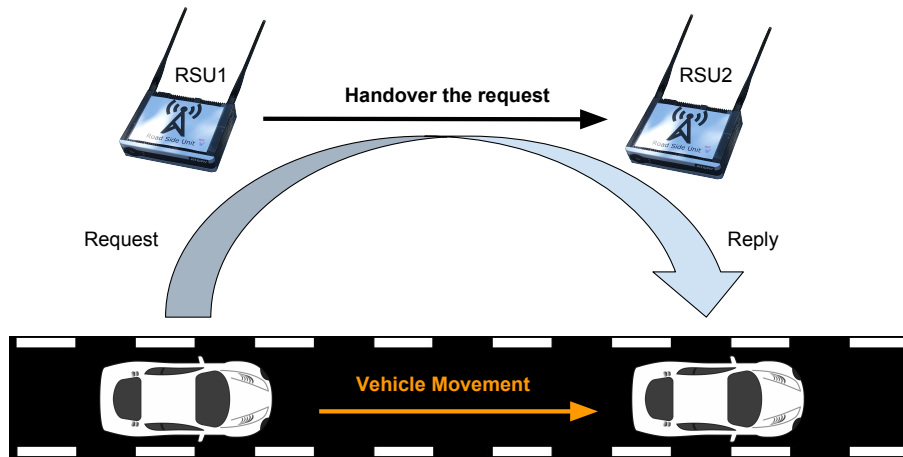


Figure 2.11 RSUs have to hand over computations to match the movement of the vehicle.

After sending a request to the RSU1, the vehicle drives out of range and connects to RSU2. To deliver the reply message, the request has to be handed over to RSU2. The reply message is delivered by RSU2.

Thus, the RSU to which a request have been issued, might not necessarily be the RSU that delivers the reply.

In the scenario, where low latency is important, it is not very likely to have such long running computations, but nevertheless, a vehicle might send a request directly before it disconnects. Thus, even for low latency scenarios it is required to handle this case. Figure 2.11 visualizes this case, where the vehicle sends a request to RSU1 and before the reply is received, the vehicle disconnects from RSU1 and connects to RSU2 due to its movement. Therefore, it is required that the request is handed over to RSU2, so that the reply can be delivered via RSU2. This process has to be transparent to the vehicle and handled by the network, since it is unknown where exactly the vehicle will be, when the reply message can be delivered.

2.7.2 Vehicle-to-Everything Communication

Beside the information from vehicles, information about traffic lights, road conditions, etc. can be shared between vehicles, and traffic lights themselves can be informed about incoming traffic. This kind of communication is called Vehicle-to-Everything(V2X) [SLY⁺16] (shown in Figure 2.12). V2X communication can be split into subcategories. It includes e.g. Vehicle-to-Infrastructure (V2I) to communicate with RSUs and traffic lights, Vehicle-to-Vehicle (V2V) to communicate directly with other cars and Vehicle-to-Pedestrian (V2P) as well as Vehicle-to-

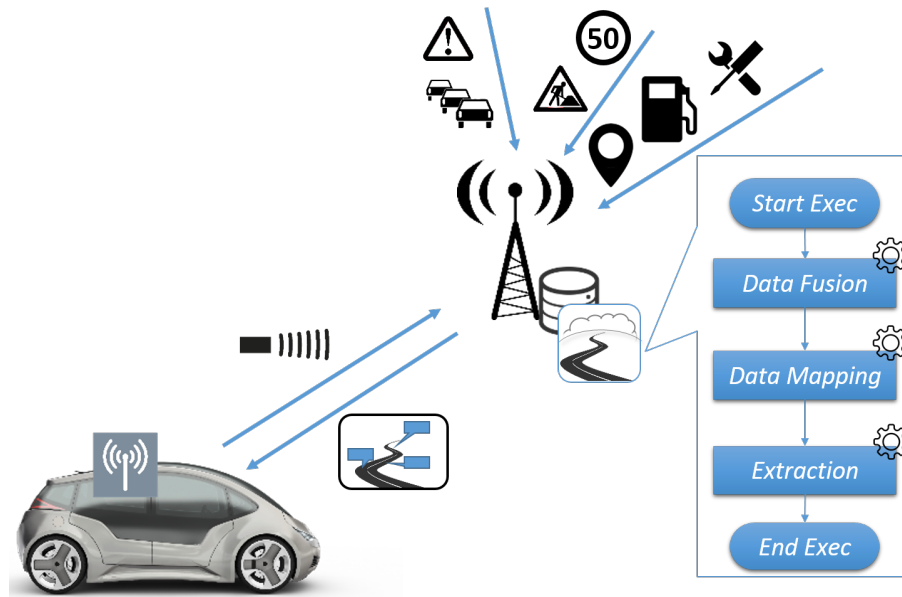


Figure 2.12 Scheme of the V2X communication (Graphic by Robert Bosch GmbH) [GWA⁺17].

The car pushes data captured by its sensors into the network, which performs data fusion operations. The car receives a virtual map about the situation on the road.

Cyclist (V2C) to communicate with vulnerable road users equipped with communication devices [CPR18]. Furthermore, Vehicle-to-Network (V2N) connects vehicles with cloud services.

On roads, not equipped with RSUs, V2V [MHA⁺15] communication can be used to share data between the vehicles. However, infrastructure based communication can deliver information in absence of other vehicles in range.

For the communication between vehicles and other road users / the infrastructure standards are defined. In Europe, this is defined the *ITS-G5 ETSI EN 302 571* [ETS08] standard. In other regions, different standards are defined, e.g. a US Federal Communications Commission (FCC) standard.

As transport layer, either the IEEE 802.11p [JD08] wireless standard which uses a 5.9GHz/z band or communication based on cellular networking (Vehicular-LTE [ACC⁺13] or upcoming 5G [RSM⁺13][MHA⁺15]) can be used.

2.7.3 Electronic Horizon

The term Electronic Horizon [REK⁺06] describes detailed information about the conditions on the road ahead. It is a virtual map containing the location and the velocity of other vehicles, information about the condition of the road and the pavement and information about hazardous conditions. Furthermore, pedestri-

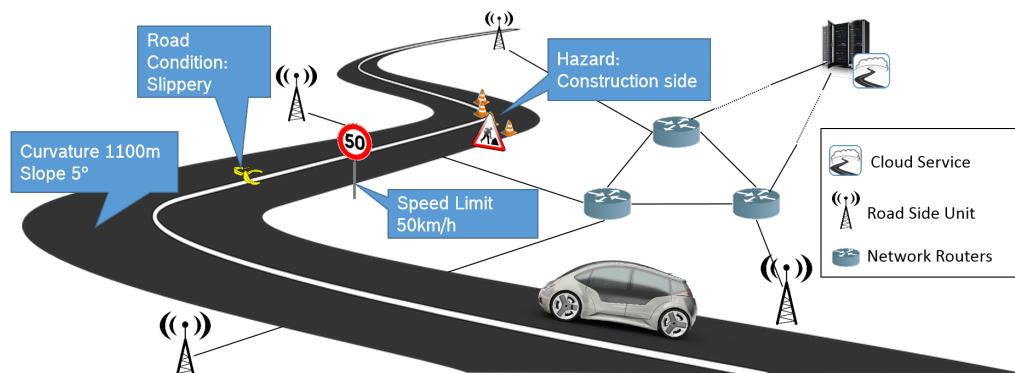


Figure 2.13 Scheme of the Electronic Horizon on a section of a road (Graphic by Robert Bosch GmbH) [GWA⁺17].

The Electronic Horizon is a virtual map containing information about the conditions on the road ahead collected by the sensors of other vehicles or by sensors of the infrastructure themselves.

ans, cyclists, animals and obstacles on the road which have been detected by sensors of the vehicles can be added to the map. Additionally, it is possible to add external data to the map, such as status information about traffic lights, gas prices, car service stations, etc.

Electronic Horizon is not required to be stored as a single map in the cloud, since it is only valid for a very short period of time and for a specific section on the road [GWA⁺17]. To increase the reliability and to decrease the latency for requests, the data for the Electronic Horizon can be distributed over different RSUs, in a way, that only the relevant data for the related road section are stored on a single RSU. A scheme of the Electronic Horizon is shown in Figure 2.13.

PART II

**Advanced Resolution Strategies in
Named Function Networking**

3

Information Used for Resolution Strategies in Named Function Networking

Data Transfers and Computations are critical for the performance of network computing and expensive if not optimal. Thus, the resolution process is the central component of NFN. Instead of only matching names for forwarding a computation, in the following we analyze the whole computation – all names in the workflow definition and even the metadata of the NDOs and the execution nodes – to achieve better execution performance.

A pure CCN request is forwarded by matching against the FIB. Thereby, the space for optimizations is rather small. Obviously, the routing algorithm creating the FIB plays an important role here. Furthermore, the matching algorithm has an impact on the forwarding, for example which entry is chosen if there are multiple matches. For NFN there are more options to influence the forwarding due to the multiple names in the IM. By choosing a different name, the computation can be forwarded to a different node using a different path and be executed on a different location in the network.

Up to this point the resolution of a computation was dependent on the first node, which chose a name that is prepended in front of the IM and the underlying CCN forwarding process. The computation was executed on a node, which had the prepended name local available either in a local connected data repository or in the CS (see Section 2.2.2). Once a name was prepended, the name was not exchanged, except when the result could not be computed on any node reached by using the prepended name, e.g. because a function was pinned to a single node.

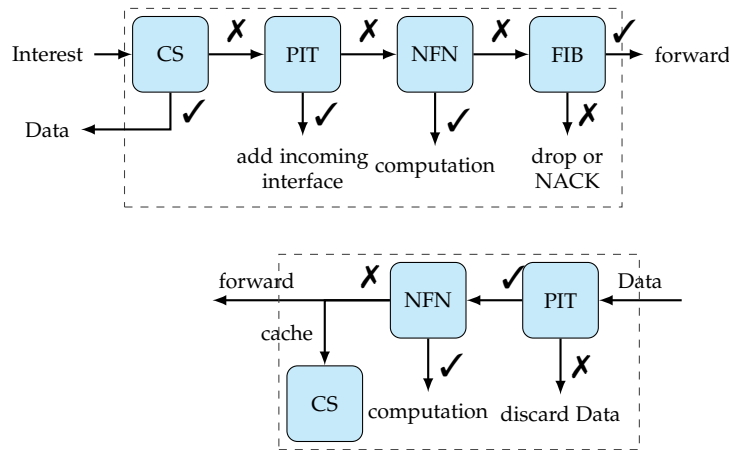


Figure 3.1 Scheme of the resolution of a computation in NFN

The NFN resolution process is injected into the CCN forwarding pipeline. The resolution strategy is a black box, which decides if a NFN IM is transformed and forwarded, just forwarded, split or executed locally.

However, choosing the first data name may not be optimal at all. For example, if the content addressed by the first data name is significant smaller than the content addressed by the second data name, unnecessary network load is created by transporting the larger data object to the node with the smaller data object instead of the other way around. Furthermore, by only forwarding the computation until the data object addressed by the prepended name is local available, the computation will not be executed in parallel on multiple nodes.

Thus, in many cases the way of resolving a computation has a huge impact on its performance and the performance of the whole network, since non optimal resolution can result in unnecessary data transfers. This causes not only in decreasing performance of the single computation, but of the entire network. Furthermore, computations can often be split up in several subcomputations, which can be distributed over different nodes and be executed in parallel.

By considering more information about the network and about the computation, it is feasible to end up with better way of resolving a computation.

A high level model of the resolution of a computation in NFN shown in Figure 3.1 is an extension of the CCN forwarding process (see Figure 2.1), where we insert a NFN handler. It is a black box, which starts, splits or forwards a computation. Furthermore, the name prepended in front of the computation can be changed. The NFN resolution strategy defines the behavior of the black box given an IM as input.

3.1 Information used for Resolution Strategies

Depending on the requirements of a specific network different information is required to find the best possible way for resolving a computation. The information can have different sources. They can be encoded in the name of the request i.e. in the computation defined in the name, they can be given by information the node holds about its role in the network, they can be given by the status maintained by a CCN forwarder and by details or metadata about the computation itself. The information is either synchronized across the network or requested on demand when it is required. This depends on the requirements of the actual network, too.

In this work, we introduce different levels of information usage for resolving a computation depending on the fraction of the available information actually used by the *resolution strategy*. The basic NFN system just uses the name of the interest itself and just chooses a name to be prepended in front of the expression. If the computation fails, another name is prepended. This is a simple *resolution strategy* with the main intention of forwarding an interest towards the input data of a *Named Function Call* first.

In the basic NFN resolution strategy (To-Data-First) a name is chosen and prepended in front of the computation. However, the FIB may already contain several names matching the expression and could be used for a wiser resolution, but this information stays unused.

The different levels of information usage are visualized in Figure 3.2 and explained in the following.

We define *Level 1* information usage as using the information that can be provided by the CCN forwarder, such as forwarding entries in the FIB, entries in the PIT or in the CS.

The *Level 2* information usage is defined by additionally using information about the mobility of a client in the network. In many networks, there are not only static clients. Some clients are mobile and they change their location and the access point they are connected to. To handle the mobility in NFN it is required to add information about the infrastructure and the mobility. This information can be used to optimize the way of resolving a computation.

Level 3 information usage is defined by using information given by the layer above, such as file size or load on the nodes or the network. This information is used to create **Plans** for executing the computation in an optimal way regarding to a given metric. Usually, CCN just forwards NDOs split into chunks,

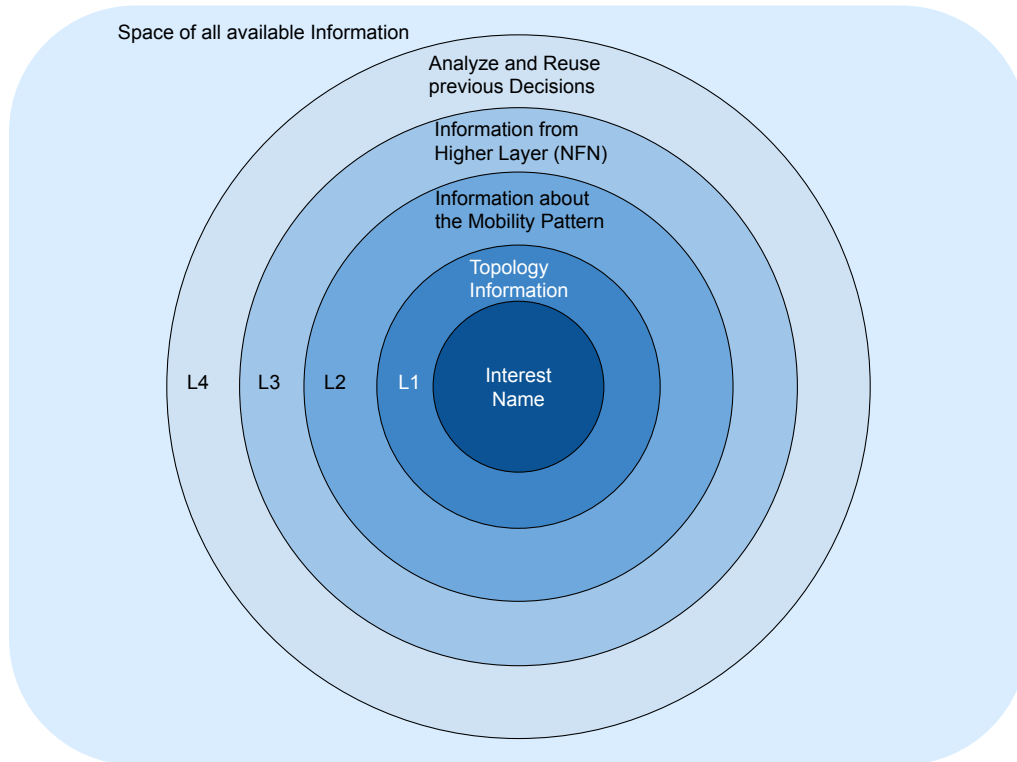


Figure 3.2 Information Levels used for Resolving a Computation in NFN.

Out of the space of all available information in the network, only a subset can be used for the resolution. Previously, only the name itself was used. But there are further possibilities such as using information from the CCN data structures (L1). Furthermore, information about the topology of the network can be used, e.g. a node knows its role in the network (L2). Information from NFN (e.g. datasize, computational load on nodes, load on links, etc) can be used to further increase the amount of available information (L3). Last, previously made decisions can be analyzed and reused for similar requests to improve and to speed up the resolutions of upcoming computations (L4).

which have roughly the same size. Thus, the CCN forwarder maintains neither any information about file-size of complete NDOs nor about the computational load, which is issued by NFN-computations. Thus, this information needs to be requested from the network before being used.

Level 4 information usage is the reusing of plans. Since computing and planning can be expensive, the planning time should not exceed the computation time. To reduce the planning time, knowledge gained by previous planning procedures can be cached for a certain time and reused. If the same computation is requested again – and there is no cached result available – probably the same plan or at least parts of the plan can be used. If a computation is similar, the network has to make a decision, if a plan can be reused, or if a new plan should be created. Therefore, it is required to compare the computation

Table 3.1 Summary of Information Usage Levels

Information Usage Level	Resolution Strategy	Chapter
L1	Map-Reduce Resolution Strategy	4
L2	Mobile-Edge-Computing Resolution Strategy	5
L3	Plan-Based Resolution Strategy	6
L4	Template-Based Resolution Strategy	7

for which the plan was created with the current. If there is enough similarity according to a certain metric, the plan can be reused.

A step further is to generalize popular plans – for example by reducing the complete data names of the original computation for which the plan was created to prefixes, which can be easily matched against new computations – in a way so the network can easily decide if the plan matches the certain computation. Thereby, by this generalization of plans templates for computations are created. These templates can be used to forward computations efficiently without creating an explicit *resolution strategy*. In other words, the network can learn how to forward computations in a given scenario, by planning how to execute a number of example computations – for example in a simulation – and then generalizing of the created plans. Sharing templates to neighbored nodes with a similar role in the network can improve the scalability of this approach.

3.2 NFN Resolution Strategies

For each defined level of information usage we create a *resolution strategy* for NFN and show the benefits.

We use the *Level 1* information to create a *resolution strategy* that optimizes Map-Reduce style computations. This is described in Chapter 4. *Level 2* information usage is used to create a *resolution strategy* for vehicular and mobile edge computing as pointed out in Chapter 5. *Level 3* information usage is used directly for creating plans as described in Chapter 6. *Level 4 information usage* is used to create more generic plans as templates. This is described in Chapter 7.

Table 3.1 summarizes the different levels of information usages and the corresponding *resolution strategies*.

4

NFN Map-Reduce Resolution Strategy

In distributed computing Map-Reduce is an important and efficient pattern for executing computations. The functional NFN workflow definition is perfect for this pattern. However, NFN is lacking the possibility to execute Map-Reduce style computation efficiently, since up to this point, the basic NFN resolution process only allows forwarding a NFN interest as it is. The computation can only be split into subcomputations if the node with the prepended name is reached and starts executing. For distributed computing it is useful to split a computation in advance, then the data path for two different subcomputations goes over different interfaces.

In the following we propose a *resolution strategy* which is designed to better match the needs of distributed computing.

For the CCN forwarding process a forwarder maintains data structures containing information about the network topology and the current state of the network. We use this information to improve the way of resolving computations in NFN. Usually, a NFN request contains multiple names. By analyzing the computation in the NFN request and comparing the different names with the CCN data structures, the decision how to resolve a computation request is made.

A common CCN forwarder has three important data structures, the CS, the FIB and the PIT. Forwarders often use a Routing Information Base (RIB) to store information about the topology such as different paths to reach a certain data object weighted by their costs. The information stored in the RIB is used to compute the FIB.

The PIT and the CS store information about the current state of the forwarder

and the FIB stores information about the network topology. For the forwarding process only the CS and the FIB are relevant. The CS is relevant, since the cached data probably reduce the time to transfer them and the FIB contains the information how to reach a single data object.

Combining this information obtained from the CS and the FIB is a way to increase the information usage compared to the basic pure name based forwarding. Instead of only considering the name prepended in front of an expression, now all names in the expression are considered. Previously, the NFN expression was transformed by the first forwarder which received a NFN request, as shown in Figure 4.1 (a). By adding additional information from the FIB the NFN expression can be transformed on any node on the route. Furthermore, not only transformations are possible, but a node can also split a computation in independent subcomputations (see Figure 4.1 (b)).

Therefore, it is required to identify independent components within the expression.

4.1 Analyzing the NFN Expression using the Abstract Syntax Tree (AST)

To analyze a NFN expression and to identify independent components within the expression we parse the expression and create an AST. An AST is an abstract structure which represents the expression. Each node in the AST represents an atomic component of the NFN expression [Wik19]. The components of the AST representing the components of the extended λ calculus are used for NFN (Equation 2.5).

In this Thesis, we define parts of the AST as shown in Figure 4.2. A subtree is a lower part of the AST. A subcomputation is a function call within a subtree. An outer computation is a function call higher in the AST than a referenced subcomputation.

The AST is a clear representation of the expressed logic without containing syntactic sugar such as brackets, semicolons or parentheses [Wik19]. Therefore, it is helpful to analyze an expression to search for optimizations. Each subtree of the AST which is either a call or a name can be mapped back to a subcomputation. Thus, during the resolution process, each expression corresponding to a subtree of the AST can be compared to the FIB separately, to increase the understanding of how the expression can be mapped to the network and to analyze all feasible possibilities how the request can be forwarded.

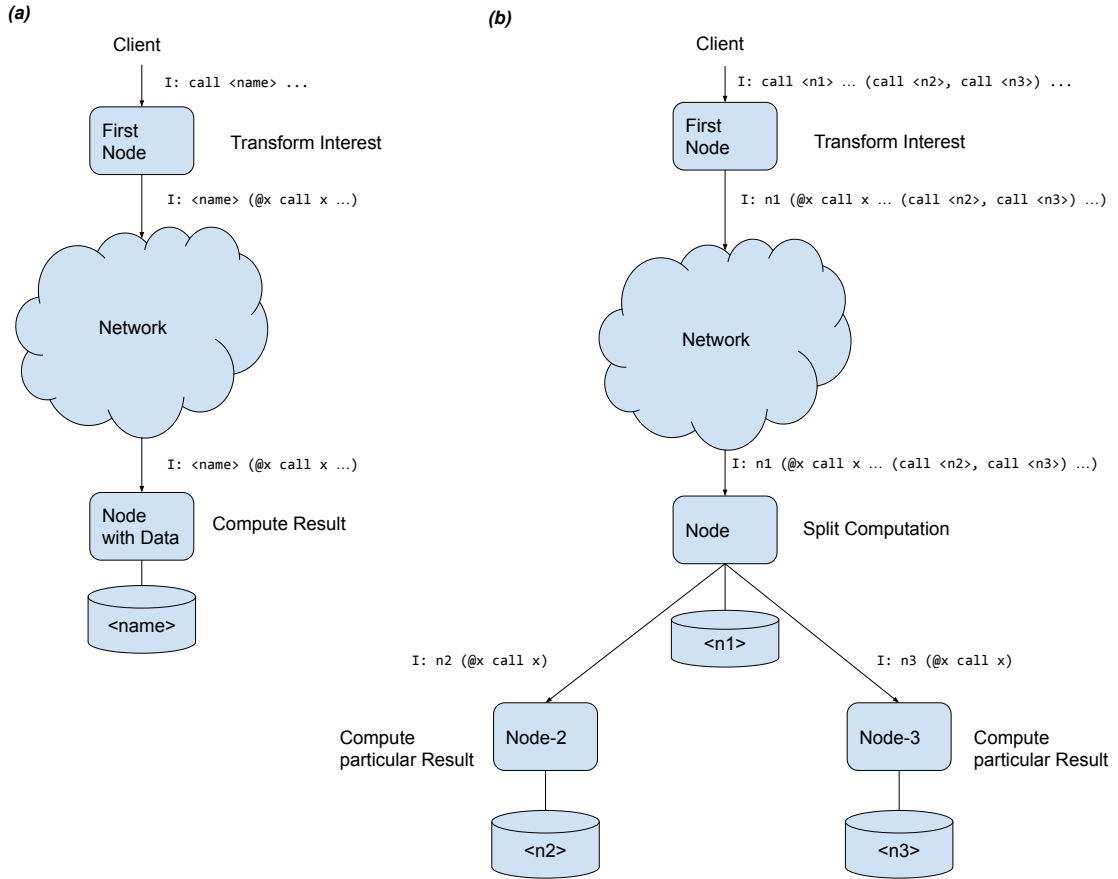


Figure 4.1 Basic Resolution of Computation in NFN (a) vs Resolution based on information out of the FIB (b).

(a) shows the Basic NFN Resolution Process, where an IM is only transformed once on the first node. (b) shows a NFN Resolution Process, where an IM is split into subcomputations by considering additional knowledge out of the FIB.

A scheme of an AST for a NFN expression is shown in Figure 4.3.

To optimize the way NFN executes computations, we look for two important patterns:

- Computations which contain independent parallel subcomputations.
- Computations which contain subcomputations that are separated from the input data of outer computations.

Since the computation workflow definition in NFN requests is based on the λ calculus, the computation is defined in a functional way and there are no side effects. In a side effect free environment, any branch of an AST is fully independent of the other branches. Thus, if a functional computation contains

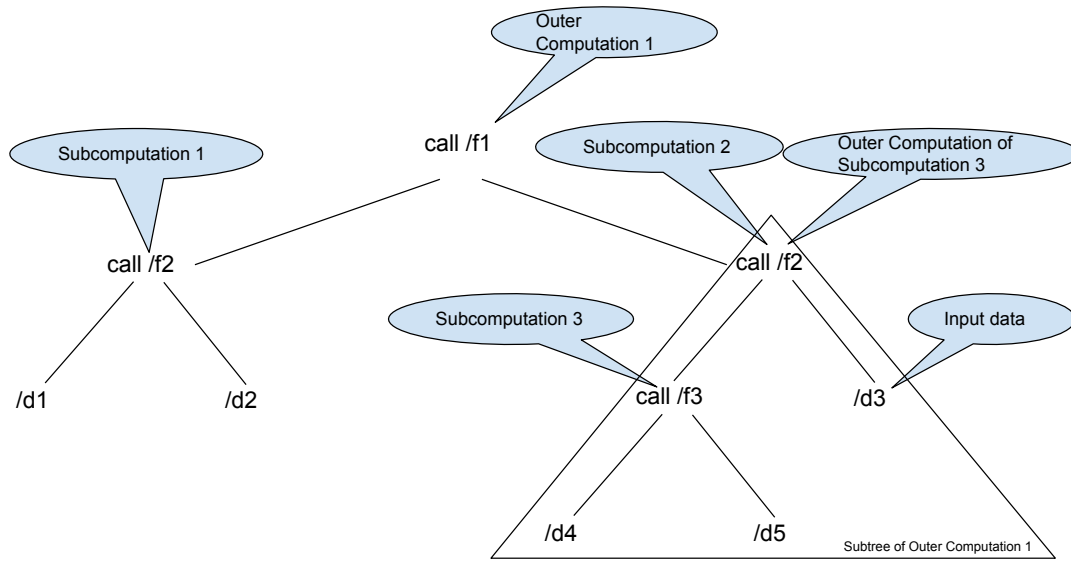


Figure 4.2 Example of an AST of a NFN expression.

We name three important components of the tree. A subtree is a lower part of the tree. A subcomputation is a computation in a subtree and an outer computation is a computation on a higher level of the AST than a referenced subcomputation.

multiple subcomputations, they can be handled independently and be executed in parallel.

For computations which consist of only a single subcomputation and input data which are independent of the subcomputation, the forwarder can decide to forward only the subcomputation and to fetch the input data or it can forward the entire computation.

An example for independent components in different branches of an AST of NFN expressions is shown in Figure 4.4.

The goal of each resolution decision is to always execute a computation more efficiently by using more parallelism or less data transfers. Thus, the decision if a computation is split into subcomputations or forwarded as it is, is critical and depends on the actual network and its requirements.

4.2 Resolution Based on the Analysis of the AST

Previously, we described how independent components can be identified within a NFN expression. In this section we will apply this knowledge to support resolution decisions.

During the resolution process a node identifies independent components. For each pair of independent components, the node decides, if the entire expres-

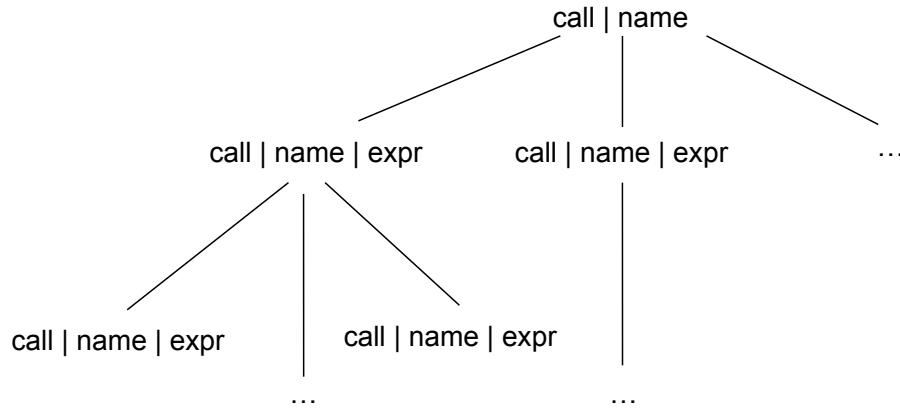


Figure 4.3 Schematic representation of an AST as used for analyzing NFN expressions.

The AST is an abstract representation of a NFN expression. It shows the structure, components and dependencies of the expression.

sion is forwarded, or if it is split into subcomputations and each subcomputation is handled separately. In case, the computation c is split into subcomputations $c_{s_1} \dots c_{s_n}$, the node which splits the computation requests the outer components for the expression to compute the outer part locally by using the results of the subcomputations $c_{s_1} \dots c_{s_n}$ as parameters, where n is the number of subcomputations.

Just because the computation c consists of several independent components, it does not mean, that it is always a good choice to split the computation into subcomputations $c_{s_1} \dots c_{s_n}$. Therefore, the resolution decision is based on the information given by the FIB, the CS and – if available – the RIB.

There are two important criteria we use to decide if a computation c should be split into subcomputations:

- For at least two independent subcomputations c_{s_a} and c_{s_b} with $1 < a, b < n$ there are forwarding rules pointing to different faces.
- Only the independent subcomputations c_{s_a} and c_{s_b} on the highest possible depth in the AST and on the same depth are considered to be split.
- Both subcomputations c_{s_a} and c_{s_b} have the same depth in the AST.

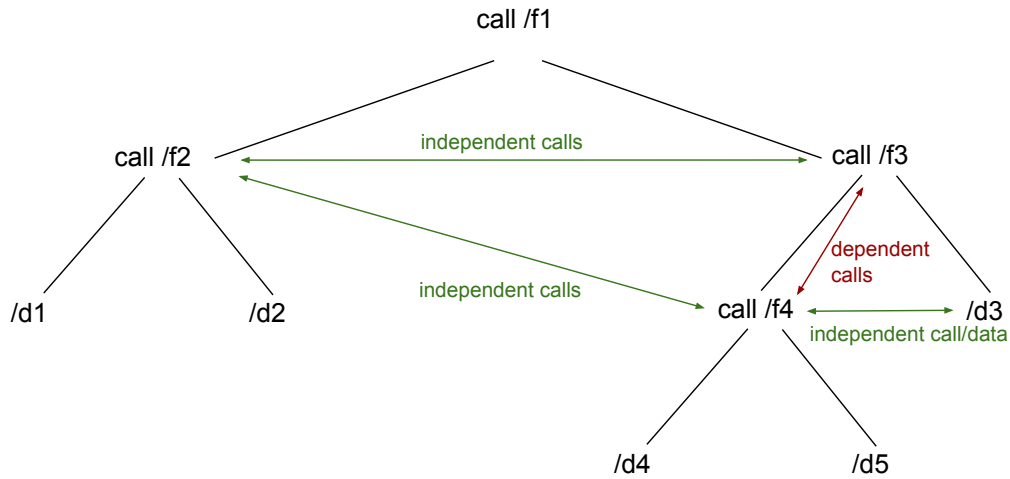


Figure 4.4 Example for independent components within the AST for a NFN expression

Components within the same branch of the AST depend on each other. Components in different branches are independent and can be handled independently. This knowledge can be used for parallelization.

$$\text{split if} = \begin{cases} \text{FIB_get_face}(c_{s_a}) \neq \text{FIB_get_face}(c_{s_b}) & (4.1) \\ \text{depth}(c_{s_a}) < \text{depth}(c_{s_1}) \dots \text{depth}(c_{s_n}) \setminus c_{s_a}, c_{s_b} & (4.2) \\ \text{depth}(c_{s_a}) == \text{depth}(c_{s_b}) & (4.3) \end{cases}$$

The first criterion is quite obvious. It does not make sense to split a computation c , when the subcomputations would be forwarded to the same face. Thus, the computation is moved as far as possible into the network before it is split up into subcomputations. Furthermore, we try to split at the highest possible depth in the AST, so that the subcomputations are as large as possible and we try to split on the same level, because if it is not on the same depth (e.g Figure 4.4, `call /f2` and `call /f3`), otherwise there would be a split possible on a higher level for at least one of the two subcomputations (e.g Figure 4.4, `call /f2` and `call /f4`).

A second possibility for splitting a computation is, if we have an input NDO ndo_1 for the outer part of the computation c on the same depth in the AST as a subcomputation c_{s_1} , to forward only a request for the subcomputation c_{s_1} and to request the NDO ndo_1 to compute the result on the local node node_1 . This only makes sense, if the NDO ndo_1 is available on node n_1 on the local CS or from a local repository. Else, this does not bring a huge benefit for the network, since the outer computation c depends on the result of the subcomputation c_{s_1} and so it is not possible to run c and c_{s_1} in parallel (see Figure 4.4, `call /f4` and `/d3`).

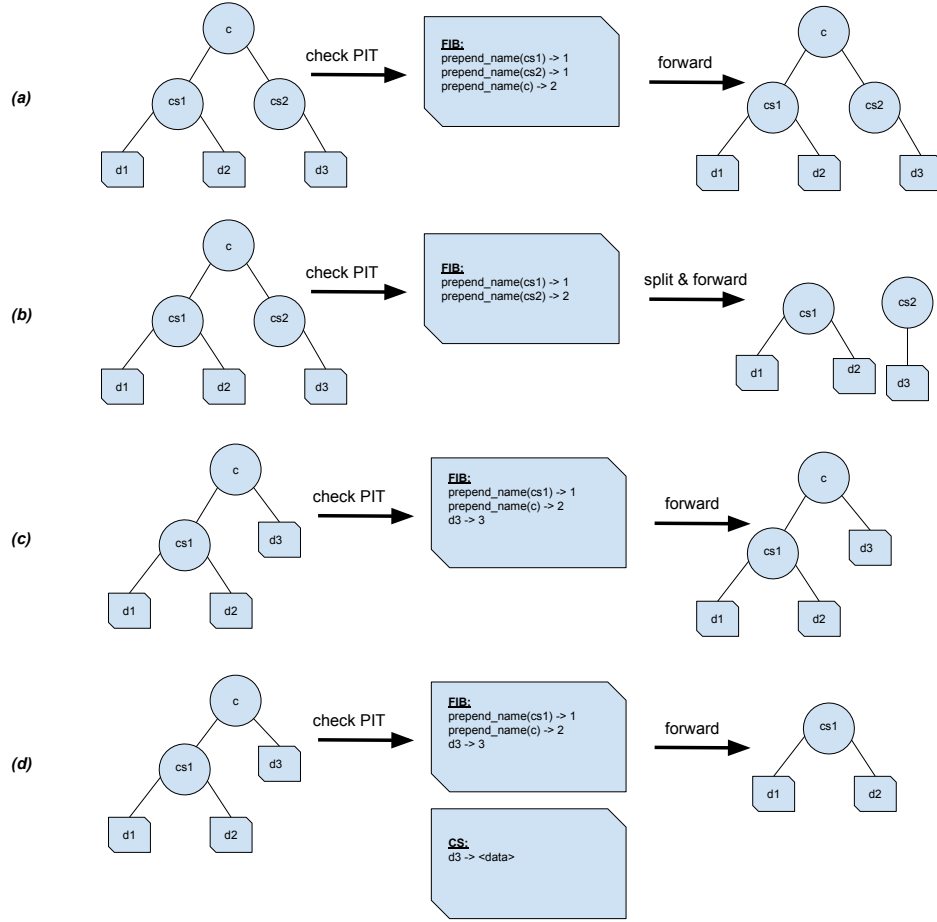


Figure 4.5 Schemes of the different cases of the resolution process using information from the FIB and the CS.

Four cases of the resolution process of a computation c . In the cases (a) and (b), there are two subcomputations $cs1$ and $cs2$ on the same depth in the AST. In the cases (c) and (d), there is a subcomputation $cs1$ on the same depth as an input data object $d3$ in the AST. In case (a) the FIB entries for the subcomputations point to the same face, thus, the interest is not split. In case (b), the FIB entries for the subcomputations point to different faces, the interest is split into subcomputations. In case (c), the input data object $d3$ is not locally available, thus, the computation is forwarded and not split. In case (d), the input data object $d3$ is local available, so that only the subcomputation $cs1$ is forwarded.

Each subcomputation is independently analyzed in the same way, as the full computation was analyzed and further splits on the same node are possible.

The schemes of the different cases of the resolution process is shown in Figure 4.5.

The *Map-Reduce Resolution Strategy* is sketched in Algorithm 4.1.

Depending on the network requirements, the decision, if an IM should be split into several requests or if it is forwarded as one IM can be adjusted.

Algorithm 4.1 Sketched Map-Reduce Resolution Strategy.

```
procedure MAPREDUCERESOLVE(computation)
  ast = createAst(computation)
  subcomps = findIndependentSubcomps(ast)
  split = false
  for all c in subcomps do
    for all d in subcomps do
      if fibLookup(c) is not fibLookup(d) then
        splitAndResolve(c,d)
        split = true
      end if
    end for
  end for
  if split is false then
    resolve(computation)
  else
    requestOuterData(computation)
  end if
end procedure
```

5

NFN Mobile-Edge-Computing Resolution Strategy

In this chapter we focus on client side mobility in NFN. Thereby, we tackle the challenge of delivering computations to a mobile node or mobile client no matter if it reconnects to a different upstream node. We specifically take a look at scenarios which benefit from edge computing, where a computation is offloaded towards nodes placed close to the client.

Usually mobile devices and nodes are placed on the edge of the network as clients and come with rather small computational capabilities or limited energy. Therefore, offloading computations into the network is very beneficial for these devices.

In modern computer networks the devices and nodes are quite heterogeneous. Depending on their task they are extremely specialized. Many of them are mobile, which opens up additional challenges for the network.

Thus, the concrete forwarding work usually depends on the movement of the clients. For NFN this means, that depending on the mobility of a node it may be required to apply specific *resolution strategies*. There are two possibilities to handle the mobility:

- Reacting to the mobility and rerouting data to the node.
- Proactive handling of the mobility and placing the data where the mobile client will be, when the data are ready.

In this thesis we only focus on handling the mobility reactively.

In heterogeneous networks different *resolution strategies* help enable the network to handle special scenarios such as mobility.

Thereby, the base stations i.e. edge computing nodes, the mobile client is connected to, handle the mobility by applying a *resolution strategy* for mobility at the edge. Only in the case, where the computation depends on data from the cloud, the edge computing node chooses a *resolution strategy* to forward the request to the core of the network. Thus, depending on the input data, the edge computing node decides to compute locally or to further offload the computation [ST18]. Meanwhile, the rest of the network executes a different *resolution strategy*, designed for cloud computing such as *To-Data-First* or *Map-Reduce*.

Handling mobility in NFN is more complicated than in CCN, since a computation can take a large amount of time and the requesting client might have changed its position far more than it would be the case for simple CCN requests.

In the following, we first discuss, how NFN can be used for offloading computations for IoT and mobile scenarios using edge specific *resolution strategies*. We start with the simple case: a scenario where the client is not mobile. Later we extend the resulting approach for mobile scenarios. Last, we discuss how to deploy different *resolution strategies* within the same network, since as we pointed out, to combine edge and cloud computing different *resolution strategies* at the edge and in the cloud are required. In addition, we take a look at another problem which generally occurs in mobile scenarios: data from mobile clients may not be available anymore, due to the movement and cannot be used. Therefore, it is required to execute computations without these data and limit the time waiting for data uploads.

5.1 Offloading Computations in Static Scenarios

In real world computer networks, not every node provides high computational capability. Especially, with the upcoming IoT, many clients placed at the edge of a network have rather little computational power. For mobile clients the battery is another constraint. Handing over computations to a specialized node can save time and energy on the device.

Offloading of computations is a typical RPC scenario. A node already holding the input data and often also the function call asks another node to perform a computation - typically because it has higher computational capabilities.

Previous *resolution strategies* focused on distributing subcomputations over the network, so, that the computation is executed on the location or close to the location where the input data were stored.

For computation offloading this is different. The nodes cannot just forward an interest message until some of the input data are locally available.

Figure 5.1 shows a flow chart of the computation offloading procedure. When a NFN edge computation node is reached, the node has to request the input data from the client and not send the full computation request back to the client, which holds the input data (this is the important difference to the To-Data-First/Map-Reduce resolution strategy). Thus, the edge computing node uses an extended *resolution strategy* which utilizes information about the interface the client is connected to. The *resolution strategy* does not forward computation requests to this interface, but executes these computations locally, so that the computation is offloaded executed. Only requests for data can be sent to the client.

5.2 Computation Offloading Strategies for Mobile Scenarios

For edge computing mobility is a more difficult challenge than for cloud computing. For cloud computing it is only required to change the path how the data are delivered. The distance to the client is basically constant. CCN natively supports client side mobility by retransmitting the IM and recovering packets lost due to the mobility from CS nearby as shown in Figure 5.2. This works, since a node is unlikely to change its location to a completely different place, but rather to stay in the same area. The forwarding itself remains unchanged, since all the network has to do is to forward the IM (for data as well as for results) towards the server.

For edge computing the situation is different. Since the executing node is placed at the edge close to the client, the client moves away from the executing node and the distance to deliver the result is increased. Since edge computing is especially used when low latency is required this can be problematic. Furthermore, the resolution process becomes more complicated, since it differs depending if the computation was started on a neighbored edge computing node or not.

For mobile edge computing and computation offloading there are two main challenges:

- The result of the computation must be delivered even if the client reconnected to another edge computing node (see Figure 5.3).

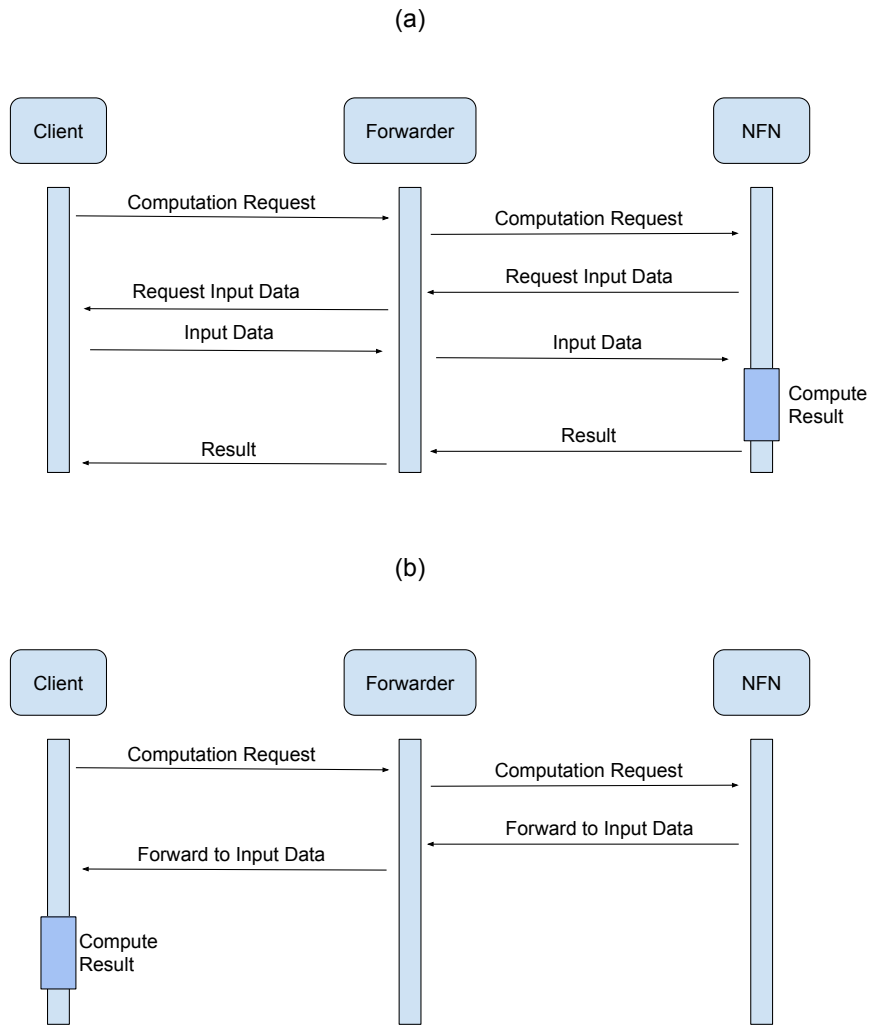


Figure 5.1 Offloading a Computation into the Network.

To offload a computation, the computing node has to request the input data from the client (a). Thereby, the *resolution strategy* of the computing node has to be different from the common case, not to forward the request back to the client (b).

- the input data must be uploaded independent of the number of reconnects occurring during the uploading process.

In the following we discuss a NFN *resolution strategy* to address this issue. First, we create a simple *resolution strategy*, aiming to pull a computation along the computation. Second, we discuss the possibility of uploading input data from the mobile node to an edge computing node for offloading a computation using multiple base stations.

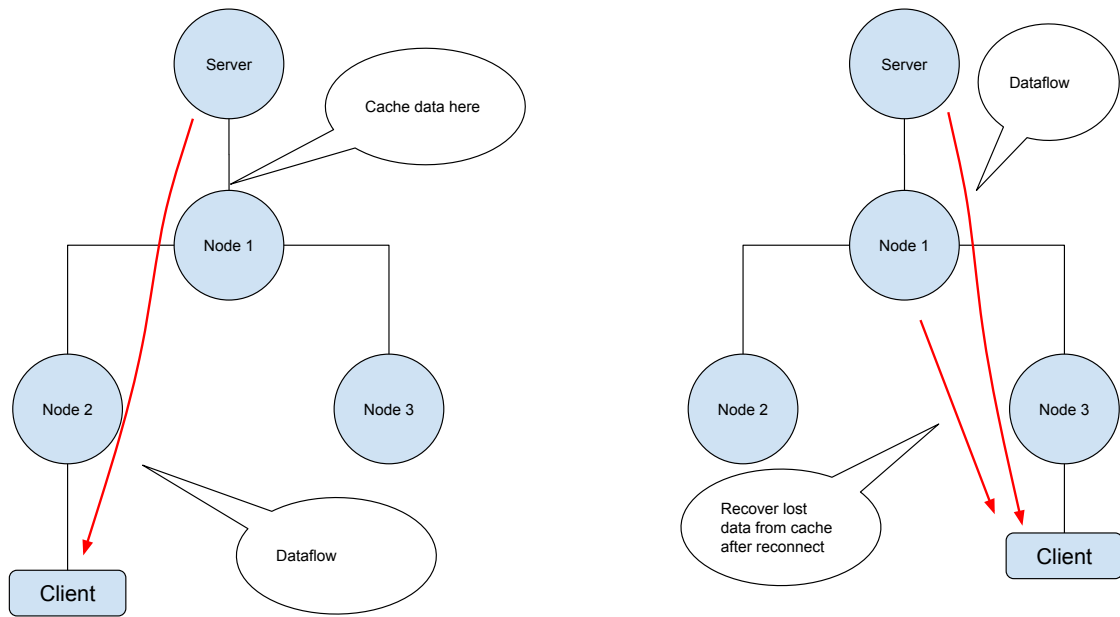


Figure 5.2 Recovering data lost due to mobility from cache.

The data flows from the Server over Node 1 and Node 2 to the client. Thereby, both nodes cache the data. When the client reconnects to Node 3, some of the packets are traveling over Node 2 but not already delivered to the client. However, these data are cached on Node 1. Thus, the client can fetch them without fully retransmitting them from the server.

5.3 NFN Resolution Strategies for Computation Offloading in Mobile Scenarios

In this section we describe a *resolution strategy* pulling the computation i.e. the result or a subresult of the computation along the movement of the client. Thus, this *resolution strategy* handles the mobility reactively. We assume a mobile scenario, where a client moves along base stations and always connects to the closest base station and the other base stations are out of range. The base stations are connected to each other with high speed connections. The mobile client simply broadcasts a message, which is received by a base station nearby. The base station checks if the same computation is already running nearby, and if this is the case, either the result or the computation, i.e. partial results are pulled to the new base station, so that the distance between client and executing base sta-

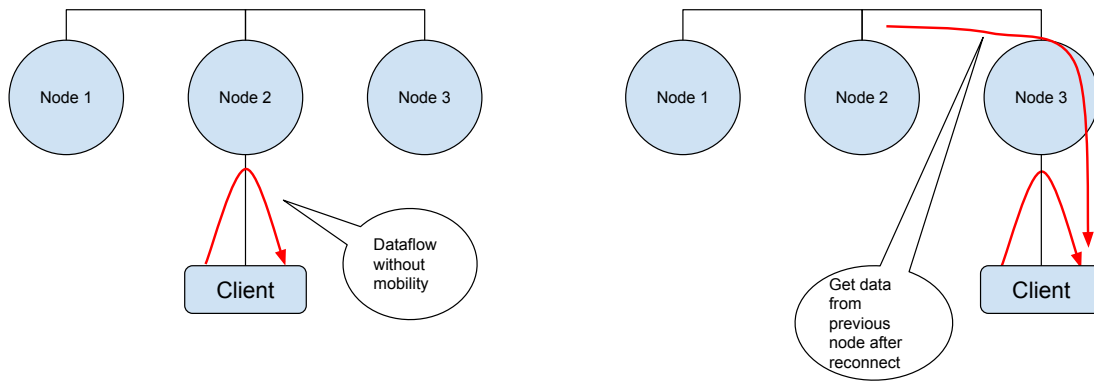


Figure 5.3 Mobility Challenge with Edge Computing.

As long as the client is not mobile, the Edge Computing scenario is simple. A computation is offloaded and the data are transmitted back to the client. In mobile case data are transmitted to Node 2 and the computation is started. When the client changes to Node 3, it receives some of the results from Node 2, the rest of Node 3.

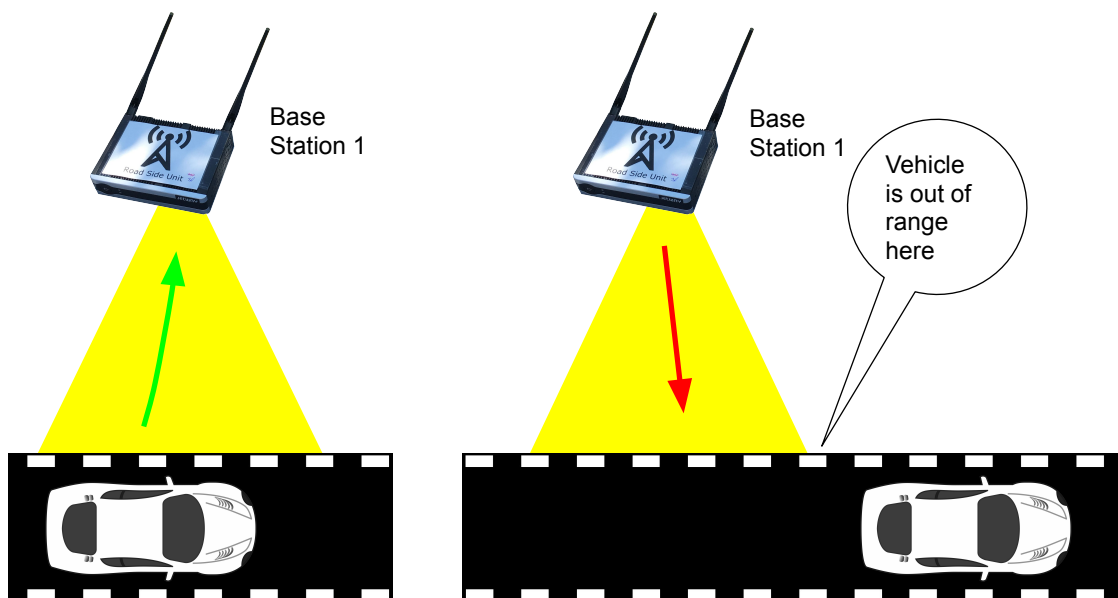


Figure 5.4 Out of Range Problem with a mobile client using Edge Computing.

While the result is computed, the car continues moving. Before the result can be delivered, the car is out of range. Without communication between the base stations, this would happen on all base stations and thus the result would never be delivered.

tion is reduced and the result can be delivered faster. If the computation would be simply restarted on the next base station without considering the already available results or partial results, on each base station the out-of range problem would occur again: the mobile client will be out of range before the result is ready, and thus it is never delivered, as visualized in Figure 5.4.

But there is also the case, where producing the result is faster than searching

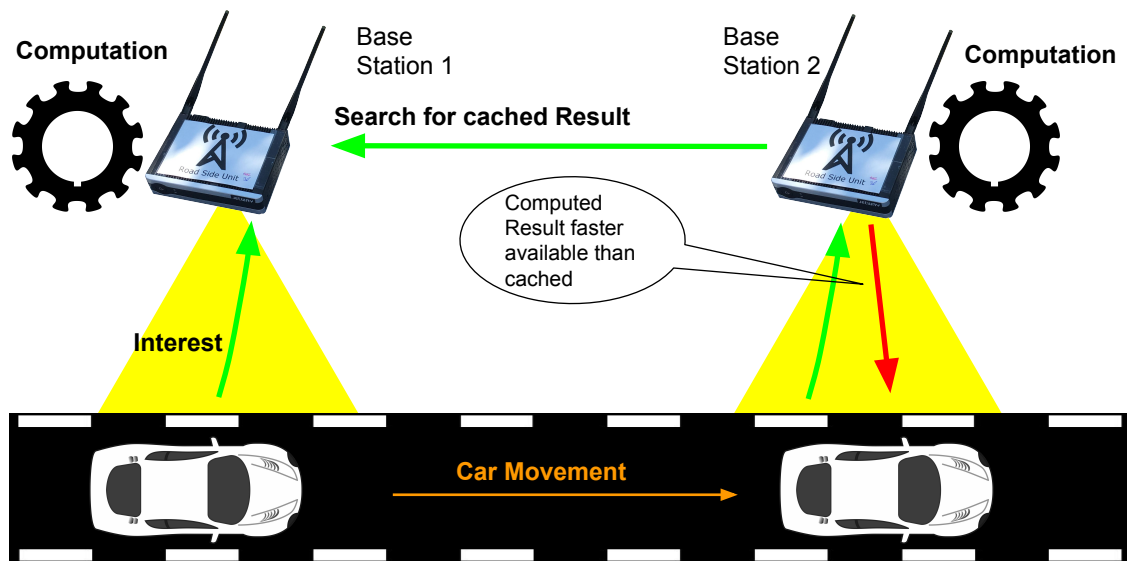


Figure 5.5 Computing can be faster than searching for a cached result.

If the vehicle transmits a short computation shortly before it goes out of range, the result can not be delivered, even if the computation is short. The next node could receive the result by searching for it or by computing it. In this case, computing is faster.

for an already computed result. Since mobile scenarios usually require low latency, it may be problematic to first search for a result which could be produced faster as shown in Figure 5.5.

Thus, for this mobile scenario we require a NFN *resolution strategy* with the following characteristics:

- Handle handover to the next base station.
- Find already running computations on neighbored nodes and use their results or partial results.
- If producing the result is faster than searching for it, then produce it.

To achieve this, we define a *resolution strategy* for mobile edge computing. This *resolution strategy* is only deployed at the edge i.e. on the base stations. In the rest of the network another *resolution strategy* is used.

In Algorithm 5.1 we present a *resolution strategy* for mobile edge computing.

The principle of the *resolution strategy* is quite simple. It starts the computation directly on the base station and in parallel it starts searching for a cached result. However, if a base station is connected to the neighbors and to an upstream node to the Internet or the cloud, it does not make sense to search for a

Algorithm 5.1 Resolution Strategy for Mobile Edge Computing.

```

c = startComputation(interest)
r = searchForCachedResult(interest)
while c.isRunning() or r is None do sleep 1
end while
if c.finished then
    return c.result
end if
if r is not None then
    c.stop
    return r
end if

```

result in the cloud. It is better to directly search only on the neighbored nodes. Furthermore, in this case we need to limit the number of forwards to avoid an IM going back over many hops along the moving path of the client. This can be done by a time to live value. As soon as a cached result is found and delivered the computation is stopped and the result is shipped to the client. If the computation finishes before a cached result is found, the computed result is delivered and if a cached result is delivered, it will be ignored. To forward an interest message to a neighbored base station, it is encapsulated using a specific prefix used for the communication between the base stations. A base station removes the encapsulation and checks if data matching the inner name is available from cache. An IM coming from a neighbored node does not trigger a new computation but only a new IM from the mobile client does.

5.4 Uploading Data for Mobile Edge Computing with NFN

Beside delivering results, it is important to upload the input data completely to offload a computation [SEM⁺19]. Thereby, the same problem as for the computation itself can occur: what happens if the node connects to another base station before the upload is completed. Restarting the upload is not an option, since the client will connect to the next base station again, before the upload is finished. This way, the computation can never be executed (as shown in Figure 5.6).

Since handling data in CCN and NFN depends strongly on the names of the data, we need to start with name space engineering. Later we use this name space for the mobile data upload using multiple base stations due to reconnects.

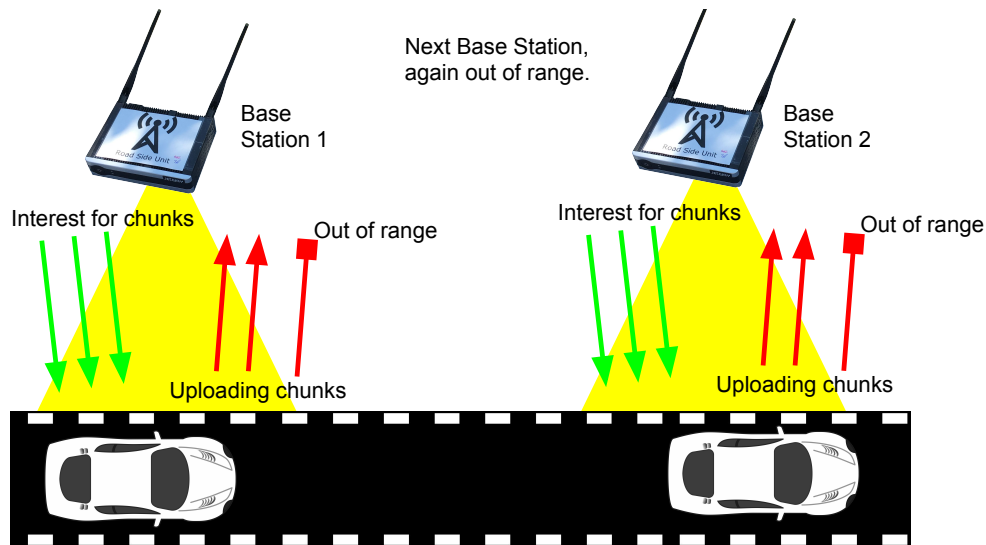


Figure 5.6 Uploading data to a single base station may not be sufficient.

If not all data could be uploaded to the first base station, because the client got out of range, that same problem will occur on the second base station, too. Without transferring data between the base stations, a complete data upload is not possible.

5.4.1 Name Space for Edge Computing in NFN

For mobile edge computing in NFN we use a specific name space to forward IMs and to decide if an IM is forwarded to a neighbored base station or to the mobile client.

To apply NFN for mobile edge computing we assume, that all edge computing nodes can reach their neighbors with an identical name. This name `/basestation` is a virtual name which is only used to communicate between the base stations (and the same for all base stations). On the other hand, the mobile client is producing data, which will be identified by a global name, e.g. data starting by the prefix `/mobile` followed by a device-id and the data-id:

```
/mobile/<device-id>/<data-id>
```

Whenever an edge computing node receives an IM containing a computation it resolves the input data. By adding a face to the mobile client with the prefix `mobility`, requests are forwarded to the mobile client. The name space is visualized in Figure 5.7

5.4.2 Data Upload using Meta Data Objects

Since the data upload in mobile scenarios cannot be performed using a single base station due to the reconnects, it is required to transport data already

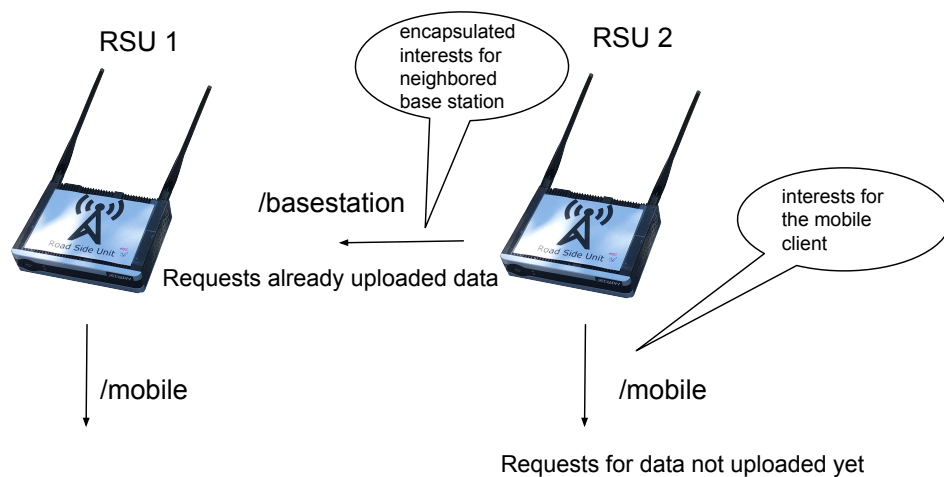


Figure 5.7 Name space for Mobile Edge Computing.

available on the previous base station to the next base station the client is currently connected to. To do this, it is necessary to understand which data are already available on a previous base station. Thus, we use a special metadata object, which lists all chunks which are available on the previous base stations or which can be made available by the previous base stations without requesting it from the client. This way, the active base station can request the metadata of the car which gives information about which chunks are required. Next, by requesting the metadata from the neighbors, the base station knows which chunks are already uploaded and can request only the missing from the car. This is visualized in Figure 5.8. The whole process of uploading data over multiple nodes is visualized in Figure 5.9.

5.5 Requirements to Combine Resolution Strategies

There are two ways to handle different *resolution strategies*:

- Deploying a different *resolution strategy* on a node depending on its tasks in the network (base station needs a different *resolution strategy* then data center node).
- A *resolution strategy*, whose resolution decision depends on the location (edge, cloud, etc.) of the node.

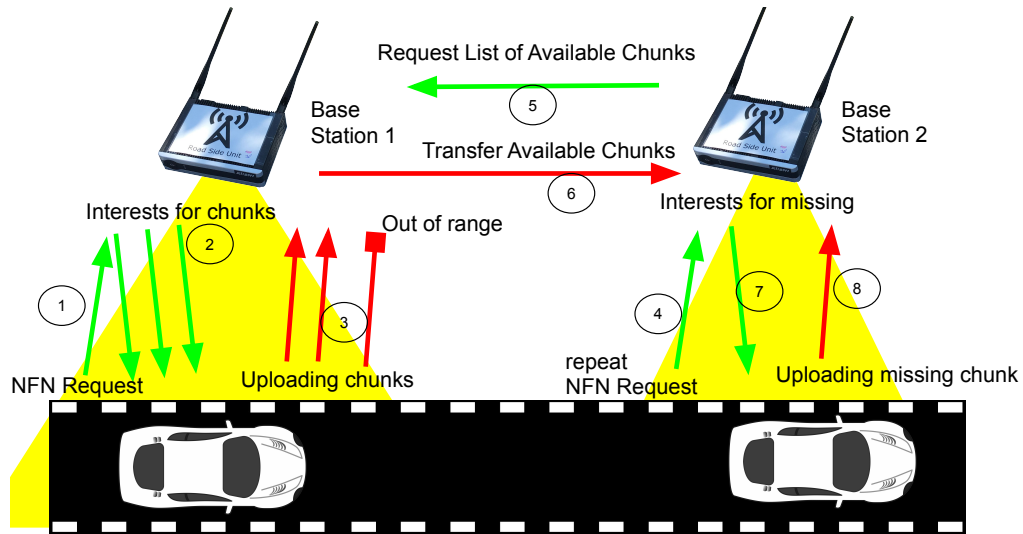


Figure 5.8 Uploading data over multiple base stations.

By transferring already uploaded data to the next base station and only requesting the missing data from the car, the data upload can be completed over multiple base stations.

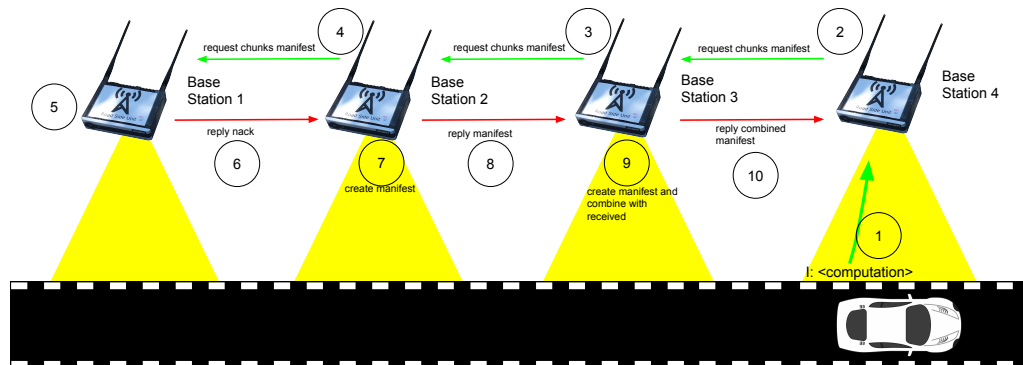


Figure 5.9 Process of fetching data from neighbored base stations.

Deploying a different *resolution strategy* on each node is efficient for the resolution progress, since there is no overhead. However, this approach is not flexible, since a node cannot change its role.

Using a *resolution strategy* which dynamically switches the resolution decision depending on the current role of a node, enables nodes to change their role.

In NFN, *resolution strategies* are designed to be deployed over an entire network. In special cases, for node mobility, computation offloading or other exceptions, the *resolution strategy* on some of the nodes has to be changed to be optimized for the special role of the node.

In general, it is possible to apply any *resolution strategy* to the nodes. Thereby,

it is important to verify that the *resolution strategies* do not contradict each other, and start to send an IM back and forth (ping pong) or to route in a loop. To prevent this unwanted behavior, different mechanisms exist. First, CCN has a mechanism to prevent looping by introducing a *nonce*, which is a random number added to an IM. If a node forwards an IM, it will check, if another IM with the same *nonce* was already forwarded and drops it, if it was.

Second, the *resolution strategy* has to take care, that it does not forward an IM back to the previous node, even if the prepended name changed. Nevertheless, if the computation encoded in the IM is split into subcomputations, the *resolution strategy* is allowed to forward the IM backwards. Both techniques are visualized in Figure 5.10.

5.6 Optional Parameters in NFN Computations

NFN is based on the idea of receiver driven communication. Thus, a receiver has to express exactly the computation for which the result should be delivered, since the name to data binding is fixed. In some mobile scenarios this is not sufficient. In case that there are mobile data producers, which provide different parameters, it is possible, that one or more data producers are not available anymore and the data cannot be delivered or not before a deadline. In these scenarios, it can be sufficient to use all input data which can be made available before a deadline, since more input data will only improve the precision or the detail level of the result.

To handle this case, we have to weaken the name to data binding: the name to data binding is fixed except for the internal communication between two hops.

The idea behind this approach is, that if a computation is issued to a base station and the base station can only collect a subset of the input data, it may be possible to continue the computation anyway. The base station will not cache a result for such an execution with incomplete input data, or only for a very short time, since the CCN name and the data behind it are not matching, due to the missing parameters. However, the result is transported towards the requester. If another base station is required to ship the result due to the client mobility, it is possible to transport the result over multiple base stations, as long as the result is not cached and not transported into other areas of the network.

The principle of variable parameters is quite simple as shown in Figure 5.11. The executing node receives an interest message, which contains a NFN expression with n parameters. Furthermore, there is a threshold t_p defined, how many

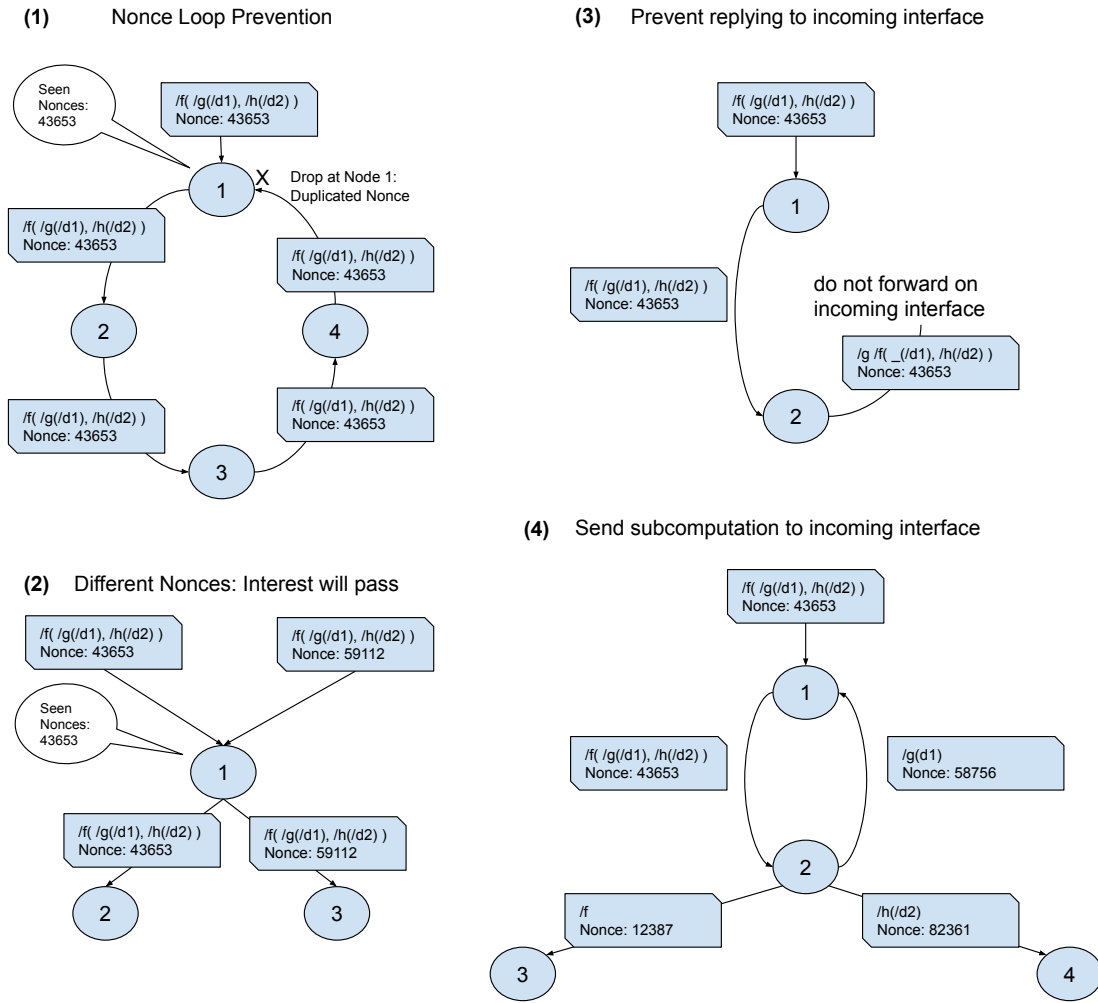


Figure 5.10 Rules to prevent loops when using different *resolution strategies* in the same network

Nonce Loop Prevention: If an IM with the same nonce appears twice, a loop is assumed and the interest will be dropped (1). If an IM with the same name appears, but a different nonce, the incoming face is appended to the PIT or the IM is forwarded, if there is no matching PIT entry (2). *Prevent Forwarding to incoming face:* If an IM is rewritten by changing the prepended name, the node is not allowed to forward the interest to the face on which the interest was received (3). If the interest is split into n subcomputations, the node is allowed to forward $n - 1$ subcomputations back to the face on which the original interest was received (4).

parameters must be available at minimum to start the computation. So, if the network can make available t_p out of n parameters the computations is started. The threshold t_p can be either defined as absolute value or as percentage of n .

The condition, if the computation can be started is checked after either a specified time or if not all parameters are either available or *nacked*.

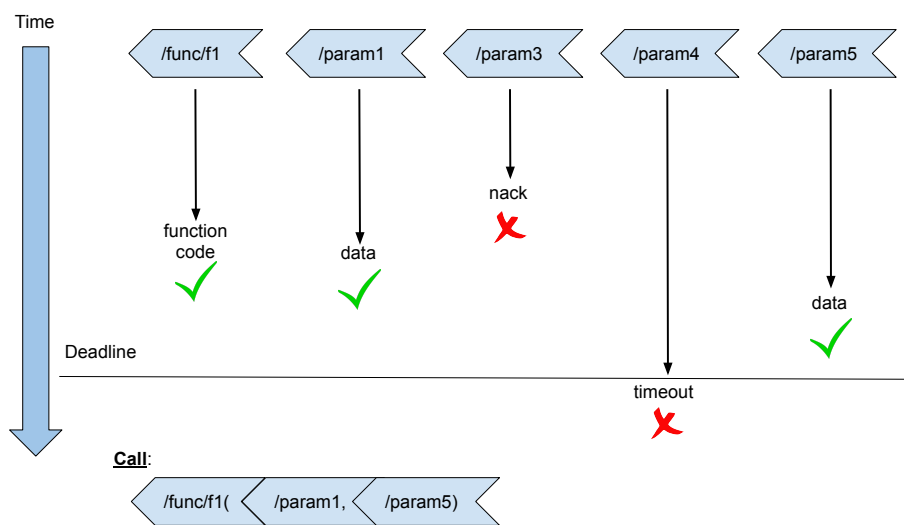


Figure 5.11 Example for Deadline Oriented Parameter Requests.

The Execution environment requests all parameters. These parameters which are available before a specified deadline are used for the function call. However, this requires, that the function call can be executed with a various number of parameters. As more input data are available, as more information will be stored in the result.

6

NFN Plan-Based Resolution Strategy

In this chapter we take a look at resolution decisions supported by information received from the network such as file size or load on the node and the network.

All previous *resolution strategies* were based on information, that were already available on the node and were designed to be fast and to find a decision with little overhead. However, by using only information, which is already available on the nodes, the *resolution strategies* rely on incomplete information. Therefore, the decisions how to resolve computations are made in regard to the actual available knowledge about the status of the network. By increasing the knowledge about the network topology and the network status, it is possible to further optimize the resolution decision [SMT19; SST16].

To increase the knowledge, the nodes in the network measure network load, computational load on nodes, file sizes etc and provide this information to the other nodes. Thereby, CCN is used as transport layer and the information about the network is encoded in NDOs. Since the network status changes over time, it is important to timestamp the names of NDOs storing this information. Furthermore, a node asks other nodes how expensive it is to forward the entire computation and execute it there.

Thus, the network chooses the best node to compute a result. To do so, it is required to define a *metric* used to compare the costs to compute on different nodes. In addition, it is required to specify, which information about the network has to be requested, so that the *metric* can be computed.

Once, the best way to execute a computation is found, a *plan* is created storing all required information to execute the computation according to the costs given by the metric.

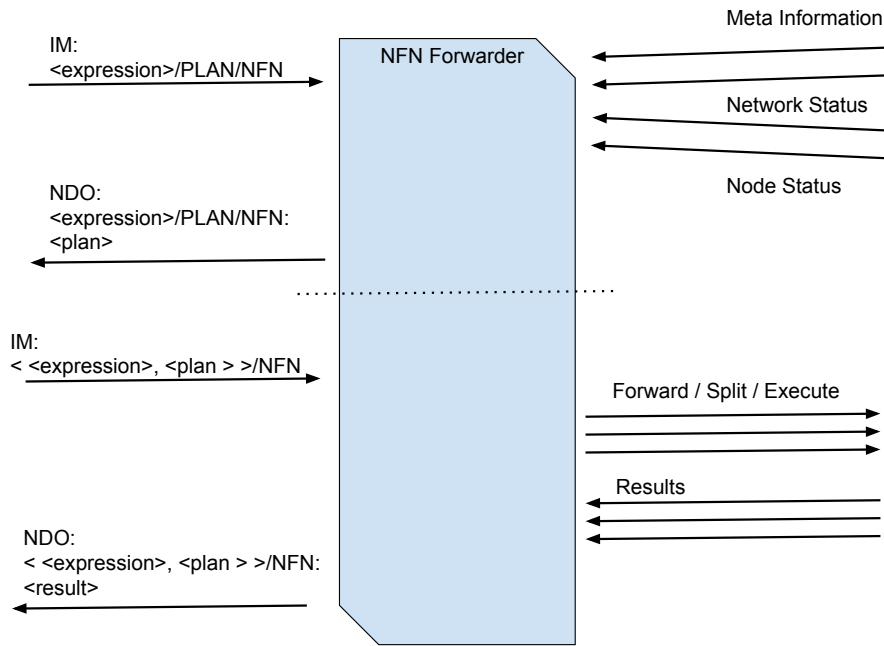


Figure 6.1 Scheme of Planning and Executing Plans

To create a plan the network will collect meta information about input parameter and the function code as well as information about the network status and the status of other nodes. If this information is collected, the network will compute the best way to execute a computation given a certain metric.

Figure 6.1 shows a scheme of the planning and execution of a NFN computation using information requested from the network.

6.1 Plans for Executing NFN Computation

After computing the best way to execute a computation in regard to a given metric, it is required to tell the nodes how the computation can be executed to reach this costs. We call a data structure, which stores this information a plan.

A Plan is a list containing an operation to be applied by each node on the route. After completing the operation the node will remove it from the plan and forward the interest. Thus, every node can simply execute the first operation defined in the plan.

We can define a plan based on two basic operations fwd and split.

$$\text{Plan} := \text{fwd}(\text{prependedName}) * |\text{split}(\text{level}, \text{Plan}*) \quad (6.1)$$

A Plan is a list of names where each node which forwards the IM according to the plan appends the next name according to fwd commands in the plan. For efficiency, it is possible to introduce a placeholder, if a node should not change the

prepended name. In case, the node should split the computation in the IM into subcomputations, the plan contains recursively a plan for each subcomputation. A plan encodes if a computation should be split into subcomputations using the split command. Thus, it is required to store, which node should split the computation, which part of the computation should be executed locally on the splitting node and how to deal with the subcomputation. The level parameter of the split command defines on which level of the AST the computation will be split. The *root* of the AST is identified with level = 0. The level parameter is followed by a plan for each subcomputation.

To execute a computation according to a plan, an IM is sent containing a tuple:

$$\text{IM} = \langle \text{expression}, \text{Plan} \rangle \quad (6.2)$$

where the expression contains the computation as defined in Equation 2.5 and Plan contains a plan as defined in Equation 6.1.

If the computation is split, new IMs containing the subcomputations are created. These IMs are only coupled with the part of the plan they require. We call the part of a plan a *subplan*.

$$\begin{aligned} \text{IM}_1 &= \langle \text{subexpression}_1, \text{Subplan}_1 \rangle \\ \text{IM}_2 &= \langle \text{subexpression}_2, \text{Subplan}_2 \rangle \\ &\dots \\ \text{IM}_n &= \langle \text{subexpression}_n, \text{Subplan}_n \rangle \end{aligned} \quad (6.3)$$

Each subplan can contain further subplans to describe the entire workflow how to execute a computation. Figure 6.2 shows the general workflow of executing a computation according to a plan.

CCN forwards IMs based on their names. For NFN, a name prepended in front of an expression is used as routing hint. A plan uses this principle by defining which forwarding node should prepend which specific name in front of the expression. This way, an IM is forwarded by using the existing forwarding system of CCN. Nevertheless, since the NFN nodes exchange the prepended name depending on the plan, the IM is forwarded to a specific node. There is one exception, in case, the network is dynamic and the route changes often, this method may not be successful. However, at least it follows a path to the name. Thus, in very dynamic networks, the costs may not be optimal anymore, when the computation is executed. Nevertheless, it is still possible to execute it.

A second possibility to define plans is to bypass the CCN forwarding system and to use the outgoing interface as NFN forwarding hint instead of the

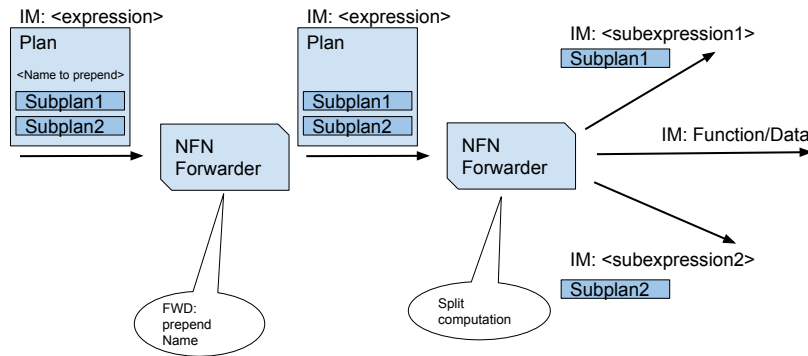


Figure 6.2 Scheme of Executing Plans and Subplans

A plan contains an instruction for every node. Here, the first node will exchange the prepended name, while the second node will split the computation into subcomputations and execute the outermost computation locally. Therefore, the function code and other input data are requested. The result is later routed back on trace of the interest, and the subresults are combined on the node which split the computation.

prepended name:

$$\text{Plan} := \text{fwd}(\text{interfaceIdentifier}) * |\text{split}(\text{level}, \text{Plan}*) \quad (6.4)$$

However, while this is more robust against changes in the forwarding tables, it does not support node mobility at all, since the forwarding rules are statically fixed by the definitions of the outgoing interfaces.

6.2 Local Named Plans and Caching/Reusing of Plans

Up to this point, a plan contained all information required to distribute a computation in the network. The plan was produced on demand for a specific computation. Unfortunately, creating a plan is computationally expensive. Furthermore, plans can become very large, so that the size of an IM is increased. Large interest messages increase the risk of Denial of Service (DoS) attacks. Caching and reusing plans is a way to reduce the planning costs. Additionally, by caching plans, the plans become part of the network instead of the IM. To put a plan into a special CS for plans on a node the plan is named. A *Named Plan* is identified by the expression encoding the computation followed by a plan tag PLAN and the NFN tag:

[<prepended name>] / <expression> / PLAN / NFN.

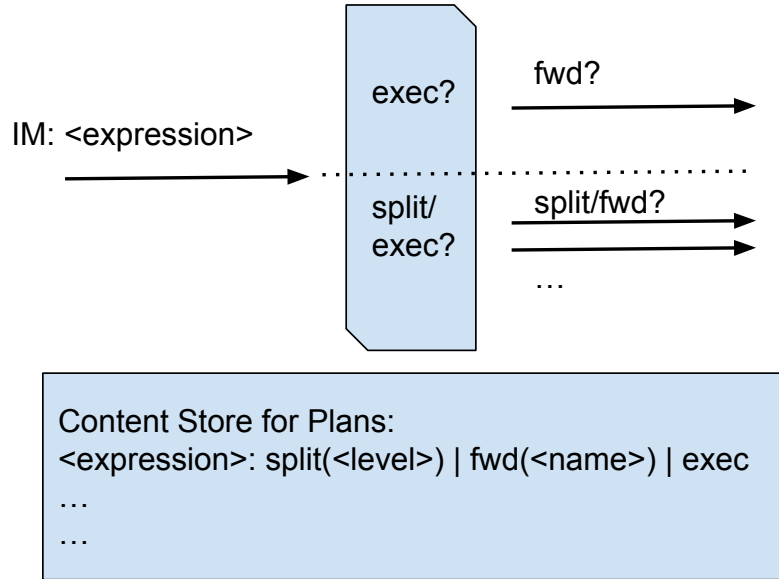


Figure 6.3 Resolution with *Named Plans*

The Content Store for Plans is mapping the expression (in the name of the IM) to instructions to be executed for the resolution process. A *Named Plan* maps the name of an expression to instructions.

If a node receives an IM, it checks if a cached plan is available and if the plan is fresh. In this case, the same plan will be reused. If an entire plan cannot be reused – depending on the information required to create the plan – some of the information may be reused. Furthermore, when using distributed *Named Plans* it is possible to update only subplans of a computation, if they are expired and reuse the parts which are still valid.

For efficiency a *Named Plan* only stores the forwarding information about the next hop. Therefore, a node may not transfer a plan to another node, since it is only valid on the node, which created it. This is also shown in Figure 6.3.

Thus, we define a *Named Plan* as:

$$\text{Plan} = \text{List}[\langle \text{expression}_n, \text{name to prepend}_n \rangle \mid \text{split}(\text{level}) \mid \text{exec}] \quad (6.5)$$

where each element of the list is stored on a node on the route. If a computation is forwarded the given name from the plan is prepended. If the computation is split, the level in the AST is given by the plan. There will be an explicit *Named*

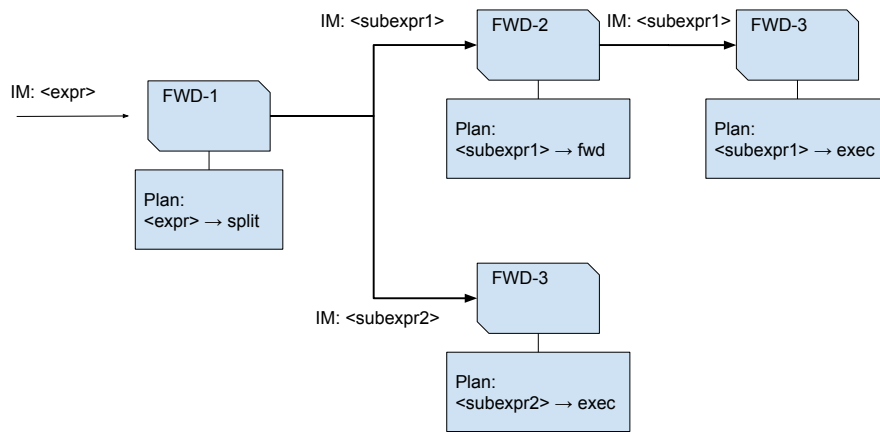


Figure 6.4 Example of NFN with *Named Plans*

When forwarding with *Named Plans*, only the information for the local forwarding decision on the actual node is stored in a plan. A *Named Plan* either contains a forward instruction, defining which name to prepend, an exec instruction to execute a (sub)computation or a split instruction to split a computation into several subcomputations.

Plan for every subcomputation stored in the node's CS for plans. If a computation should be executed, the plan expresses this using the *exec* command.

An example of how an IM is forwarded within a network is shown in Figure 6.4. A distributed plan storing only the relevant information for a single forwarding decision on the forwarding node can be easily reused, even for subcomputations since they are encoded in separate *Named Plans*. This is possible, because if a computation is split into subcomputations, the nodes handling the subcomputations are only storing the required information about the subcomputations and the plans for subcomputations are identified by their names. Having computations consisting of many subcomputations makes it likely that parts of a *Named Plan* can be reused.

6.3 Requesting Meta Information and Creating Plans

To plan the best possible execution for a computation all required information must be collected. Next, this information must be combined to find the plan with the lowest costs.

The planning procedure can be briefly described as:

1. Collecting meta information about the computation and the input data at

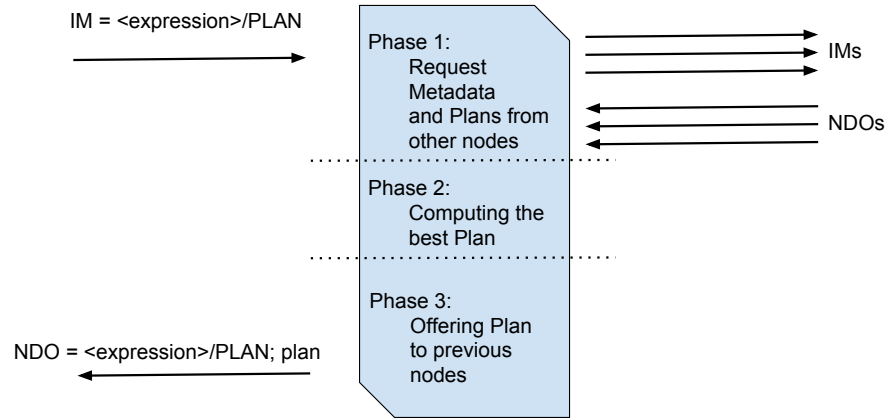


Figure 6.5 Scheme of Creating Plans

To create a plan, each node which is considered to execute the computation will compute the costs for executing the computation locally and all this information combined will give the best plan how to execute a computation.

each node.

2. Choosing the cheapest option at each node.
3. Collecting all options to create an overall execution plan.

A scheme of the plan creation process using a plan is shown in Figure 6.5

6.3.1 Metrics to Compute Optimal Execution Location

To create a plan for executing a computation at low costs, it is required to determine the costs of executing a computation. This is usually done by using a metric i.e. a cost function to map the hard to describe costs to a single number which is comparable:

$$c(x_1, \dots, x_n) \rightarrow \mathbb{R}, \quad (6.6)$$

where n is the number of input parameters and the cost function c maps the input parameters to a real number. The cost function can include various input parameters such as content size, load on nodes, the load on the network, the expected execution time of a function, etc. The input parameters are chosen depending on the requirements of the actual network and the cost function c weights all parameters depending on how important they are.

An example for a cost function is:

$$c(fs, cl, mb) = \sum_{x=1}^n \frac{2}{3} \cdot \frac{fs_x}{mb_x} + \sum_{y=1}^m \frac{1}{3} \cdot cl_y, \quad (6.7)$$

where fs is the file size, n is the number of input files, mb_x is the minimum bandwidth to fetch the file fs_x , cl_y is the computational load on the y – th executing node and m is the number of functions involved in the function call. In many scenarios, it makes sense to weight the file size higher than the load on the nodes, since the load on the node can quickly change.

6.3.2 User Defined Metric

Up to this point, we only considered a static metric deployed all over the network. In the following we consider a planning request to be a tuple of a computation expression and a metric:

`<expression; metric >`

There are basically three possibilities how to define a metric for this tuple:

- An expression, which describes, how the metric is computed.
- A NDO which contains a function how to compute the metric.
- Changing parameters for a given metric.

Using a user defined expression, it becomes possible for everyone in the network to define their own metric. However, the expression still relies on information that can be delivered by the network. Thus, only information, which can actually be delivered by the network can be used and having a user defined metric per request undermines the possibility of reusing plans.

Therefore, a more practical solution is to define some “named metrics” a user can choose from. Thus, the number of metrics is limited and the chance of a cache hit is increased. A predefined metric enables the developer of the network to know in advance which information can be requested for the planning process and thus it is possible to provide all information which may be required.

A third approach is defining the parameter for the metric. Assuming a metric is a linear combination of the various parameters, users can easily adjust the metric to their requirements:

$$\text{costs} = \sum_{x=1}^n w_x \cdot c_x \quad (6.8)$$

where w_x describes the weights and c_x the cost value given by a parameter. In this case, the metric is defined as a list of values, which are the weights w_x :

$$\text{metric} = \langle w_1, w_2, \dots, w_n \rangle . \quad (6.9)$$

The big advantage of the parametric based metric is, that the network can cache the individual values of the file size or load and reuse them to compute a new plan, since only the weights of the parameter are changed.

6.3.3 Requesting Information about the Network Status

It is required to fetch metadata information about NDOs, function code or the status of nodes from the network. To stay in the context of CCN this information will be encoded in NDOs.

There is a difference between different types of meta data. There are meta data which can change over time such as the load on the network or the load on the nodes, while other information is quite static such as the size of a NDO or the expected time to execute the function code.

It is required to set the cache time within the meta data of a NDO storing non static information to a small value or disable caching for this data at all to avoid shipping outdated information. Storing non static information under certain names violates the strong name to data binding concept of CCN. However, this information is not exposed to the user but only used internally.

To name the meta information about a NDO we add an additional name component at the end, which tags that NDO as meta data information:

```
<ndo name>/METADATA.
```

Beside the information stored explicitly in meta data objects, some further information about the NDOs can be stored in the manifest of a chunking protocol such as FLIC (see Section 2.1.4).

Requesting information about NDOs in the context of CCN is quite simple since the information can be bound directly to a name and data tuple. Requesting information about the load of a node or of a link is more complicated, since in CCN there is no node identifier.

To circumvent this issue, we take advantage of the recursive plan structure which includes subplans. A subplan with the instruction to execute the entire computation considers the computational load on the executing node in the costs of the plan. Since a plan is a combination of the subplans and the costs of a plan are the sum of the costs of the cheapest subplans, the meta information about the load of a node will be already included into the overall costs, when the plans are combined (see Figure 6.6, Subcomputation 1 i.e. Subplan 1a and 1b.). Furthermore, it is possible to consider the network load and the bandwidth

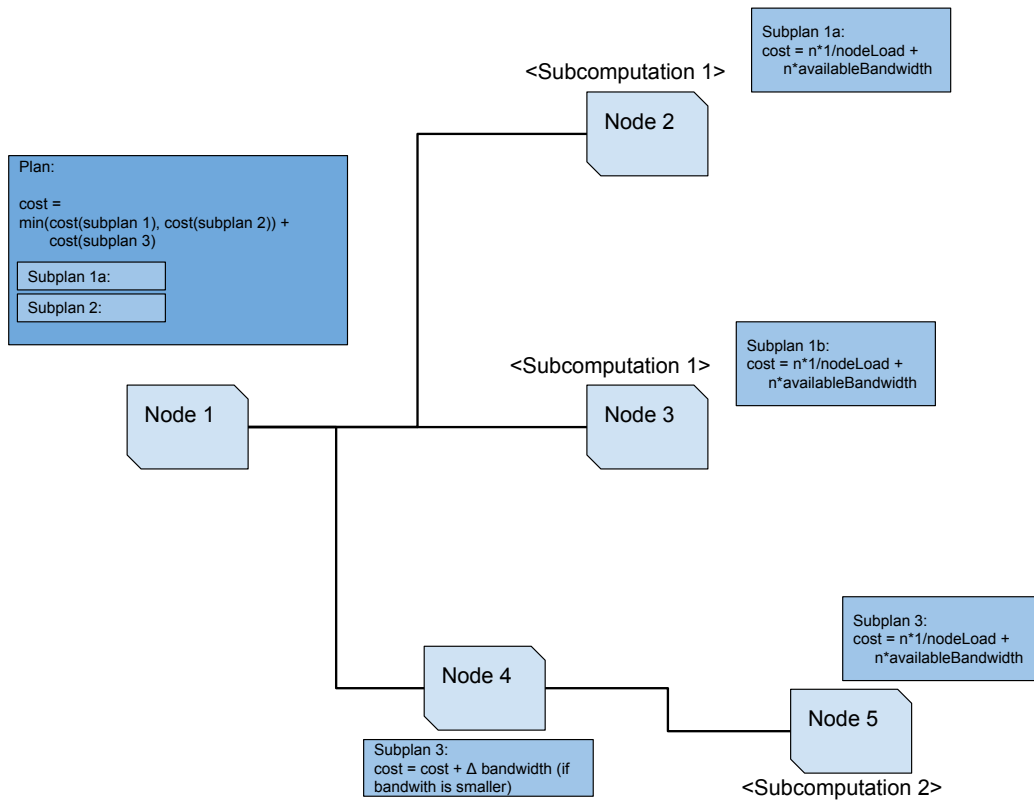


Figure 6.6 Combining Subplans to an overall Plan

The subplans which contain the instructions to execute the subcomputation will consider the load on the node into the costs. Node 2 and Node 3 are candidates to execute Subcomputation 1. In the overall plan, created on Node 1, the cheaper option is chosen, in our case Node 2, assuming the costs of Subplan 1a are smaller than the costs of Subplan 1b. Subcomputation 2 is executed on Node 5. However, the costs of Subplan 3 are increased on Node 4, in case the available bandwidth on the link between Node 1 and Node 4 is smaller than the available bandwidth on link between Node 4 and Node 5.

of links, when combining the different subplans on a node. Thereby, the transporting costs will be stored separately to the rest of the costs, so that they can be changed in case there is a link with less available bandwidth on the way (see Figure 6.6, Subcomputation 2 i.e. Subplan 3). This way, not only the issue to address nodes is eliminated, but also the problem of caching non-static meta information is circumvented. *Plans* become the only NDOs without a static name to data binding.

6.3.3.1 Resource Allocation

When nodes provide information about their load or the network load, this information is only valid for a very short time and can change anytime. De-

pending on the scenario and the number of requests the planning system uses information which is not valid anymore when the actual computation is executed. Therefore, the node can allocate resources and hold them available for the computation to be executed. However, the planning process involves nodes, that will not be involved in the final execution process. Therefore, just holding resources available on a lot of nodes is not efficient at all, even if they hold these resources only for a very short time. Our solution to this is, to tightly control which resources are allocated. When choosing the cheapest option the node offering this is informed about the choice, so it can allocate the resources. This is done by sending an IM and confirming the allocation with a NDO similar to requesting the meta information.

6.3.4 Creating Plans

When creating a plan each node will compute the cheapest way to execute the computation locally given a metric to score the costs of a computation.

Our planning algorithm is a distributed algorithm, where every node searches for the lowest costs with itself as starting point. Each node collects the minimum costs for forwarding the entire computation and the minimum costs for all sub-computations as well as for computing locally.

The planning algorithm can be briefly described as:

1. Selecting possible subcomputations and possible options for forwarding on each node.
2. Choosing the cheapest option at each node.
3. Collecting all options from all nodes to create an overall execution plan.

In the following we will go through all parts of the planning algorithm.

6.3.4.1 Selection of Possible Subcomputations and Options for the Resolution of Computations

In general, it is possible to run a computation on any node in the network, as long as the function code and the data are mobile.

Since planning algorithms are often NP hard [FLRK80], it is important to carefully select the nodes, which are considered by the planning algorithm. Therefore, a planning request is not simply broadcasted to every node in the

network. Given the names in a computation request and the FIB, the network can restrict the number of nodes to be considered for executing a computation.

Selecting the subcomputations for planning can be divided into four phases, as also shown in Algorithm 6.1:

1. Parsing the expression and creating the AST.
2. Finding the subcomputations with the AST.
3. Choose which names should be prepended in front of each subcomputation.
4. Filter out duplicated subcomputations or subcomputations which do not create additional knowledge.

Algorithm 6.1 Choosing the possible subcomputations for the planning procedure.

```

ast = parseExpression(<expression>)
subcomps = getSubComps(ast)
ims ← [ ]
for all sc ← subcomps do
    v = prependNames(sc)
    ims.add(v)
end for
ims = filterSumComps(ims)
return ims

```

To note here is, that the list of subcomputations contains the actual computations, since simply forwarding the entire IM is a possible decision, too.

Once, a node receives a request to plan the execution of a computation, it will start analyzing the expression which encodes the computation. First, the expression is parsed and the AST is created. Next, the AST is traveled to find all function calls, since each function call is a possible subcomputation.

To reduce the number of possible subcomputations in which the computation could be split off, we can limit the depth when searching for subcomputations in the AST per node. If the first node will only search in the depth of N , the next node will analyze the computation as well as all forwarded subcomputations, which means, the next node will look into the depth of $2 \cdot N$, while it can skip all subcomputations already created by previous nodes, which the node can detect by checking the PIT for all IMs it received. Therefore, the load of checking the

possible subcomputations during the planning process can be distributed over the network.

Since, each subcomputation may consist of multiple names, too, there are multiple variations of each subcomputation by prepending a different name. However, there are limited possibilities for forwarding. For all variations of the same subcomputation which would be forwarded to a single face, only one is chosen.

Moreover, we can apply other techniques to reduce the planning amount. For example, we can apply an AST analysis as described in Section 4, where we only consider independent subcomputations which can be forwarded towards different faces. For the planning system, this means, if a node can rule out, that a computation can be forwarded differently than it already would be, it skips the planning process and directly forwards the IM.

6.3.4.2 Choosing the cheapest option at each node:

As soon as a list of all subcomputations to be used for the planning is created, the second phase of the planning process is started. The goal of the second phase is to collect all required information for a local forwarding decision:

How to optimally handle the interest message, if it is received on a specific node.

For this decision, it is required to compute the costs of all subcomputations and to find the best combination.

To do so, the node will request the costs of resolving each subcomputation as well as of computing it locally. To compute the costs of computing locally, the metadata for each input NDO are requested. The metadata for a NDO are stored either within the manifest or can be explicitly requested by addressing the NDO tagged the postfix PLAN.

`<ndo-name>/PLAN`

This way – using the same method as described in Figure 6.6 – it is possible to include the bandwidth of the links into the costs. This is a better measure, since it tells how expensive it is to transport the input data to the requesting node.

The costs of each selected subcomputation is requested the same way as for the NDOs:

`<subcomputation-expression>/PLAN`

As soon as all required information is requested, the node searches for the cheapest combination of all possibilities. To find the cheapest combination we use a *bottom-up* algorithm, which combines the cheapest path for each subcomputation in the AST.

Algorithm 6.2 Choosing the best combination from each possible plan combination. The functions `planFWD` and `planLocal` are lookups, since these data have been requested previously.

```

procedure FINDBESTCOMBINATION(ast)
  if ast is funcCall then
    for all p in ast.params do
      plan_local = findBestCombination(p)
      plan_fwd = planFWD(p)
      if cost(plan_local) <= cost(plan_fwd) then
        plan.append(plan_local)
      else
        plan.append(plan_fwd)
      end if
    end for
  else
    plan.append(planLocal(p))
  end if
  return plan
end procedure

```

The previous requested data gives the required information about the costs of resolving a (partial) computation, while Algorithm 6.2 compares these costs with the costs for computing the same (partial) computation locally.

Figure 6.7 visualizes the workflow of finding the best combination of the planning algorithm.

6.3.5 Clustering Regions and Prefixes

Computing the optimal costs require a huge amount of computational power. Even if we already reduced the computational load by decreasing the number of requests and possible subcomputations, especially, for computations containing a large number of subcomputations, the load is still high.

In the following, we discuss a further optimization technique: clustering of prefix regions. Therefore, we assume, that the prefixes are not randomly distributed over the network, but they correlate with the physical location.

Relying on this assumption, it is possible to introduce regions in the network. A region is defined by a prefix and is considered to be a single node for the

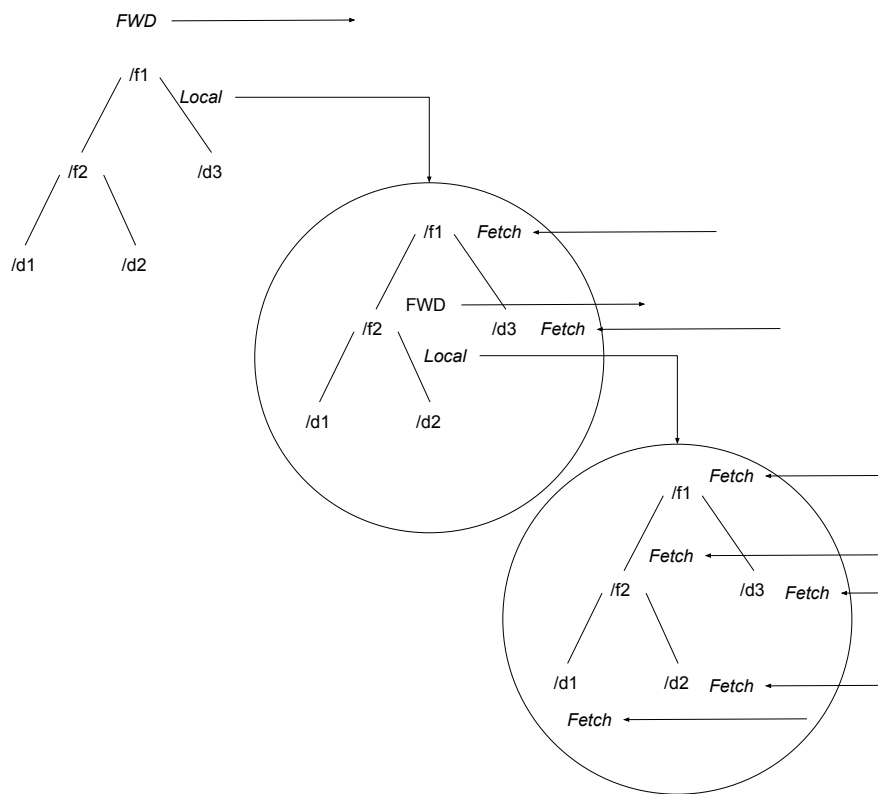


Figure 6.7 Finding the Best Combination of Forwarding and Local Computations

Compare the costs of either forwarding everything, forwarding only a sub-tree or fetching everything and computing local.

planning procedure (see Figure 6.8). This way, the number of nodes eligible during the planning process is reduced i.e. the logical topology is simplified.

Furthermore, if multiple parameters have a common prefix, which is also a prefix of a region, it is considered as the same name during the planning procedure.

As soon as a planning request for a subcomputation reaches the region with the corresponding prefix, the full name will be considered, to find the best node to execute the computation within the region.

In some special cases, this does not lead to perfect planning results, for example in the case, when the execution node is quite close to the border of a region. However, since the planning overhead is reduced, this may be a fair price to pay.

Algorithm 6.3 shows a way to create clusters. Each prepended name of the possible subcomputations is checked, whether it has a common prefix with another prepended name of a possible subcomputation. Thereby, a cluster can contain a subcluster, which consists of at least two prefixes within a single cluster which is more specific than the other prefixes. A subcluster is considered to

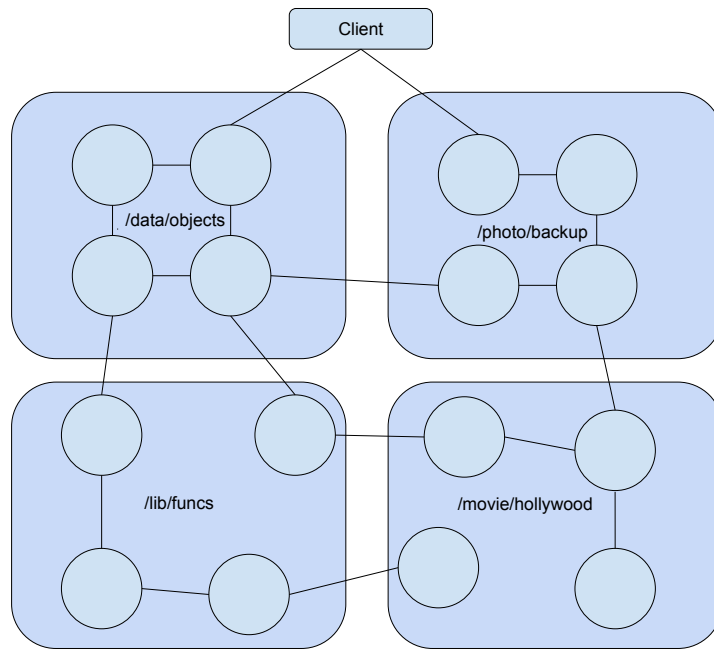


Figure 6.8 Clustering of Prefixes

Nodes holding data with a common prefix can be considered as a single node during the planning procedure, as long as the prefixes and the location of the nodes correlate.

Algorithm 6.3 Creating Clusters of Prefixes to optimize the planning process.

```

for prefix in subcomps.prefixes do
  while prefix do
    for p in subcomps.prefixes do
      if p == prefix then
        continue
      end if
      if prefix is_prefix_of(p) and p is not in clusters then
        clusters.add(prefix,p)
      end if
    end for
    prefix.remove_last_component()
  end while
end for
return clusters

```

be equivalent with a cluster. However, one can define a minimum cluster size. If a subcluster should contain at least three different names, and there is one with only two names in it, it will be merged into the outer cluster.

After creating the clusters, all subcomputations are cut out, if another computation of the same cluster is already being forwarded (see Algorithm 6.4).

Algorithm 6.4 Reducing the number of requests for meta information based on clusters.

```

for sc in possible_subcomps do
  comps_in_cluster = cluster.get_comps_in_cluster(sc)
  if None Of comps_in_cluster is in requests then
    requests.add(sc)
  end if
end for
return requests

```

6.4 Reusing of Plans

In Section 6.2 we described distributed *Named Plans*. Since the time for creating plans is high, reusing plans for entire computations or at least for subcomputations improves the performance. Reusing a cached plan is straight forward. If there is a valid cache entry in the CS, the cached entry will be used. Usually, a cache entry is valid for the time it is cached, and deleted if it is not valid anymore. Since the data used for the planning are often very dynamic and since the planning process takes some time after which the cached data might be invalid, it makes sense, that the planning algorithm itself checks the timestamps of cached plans.

In general, there are factors, which have a higher influence than others. For example, small NDOs influence the overall costs less, even if they have to be fetched over a low bandwidth connection. Since the knowledge about the data size is already available, this knowledge can be used to apply a plan to a computation, which is not an exact match to the name of the plan.

To figure out, if a plan can be reused, even if there is no exact match, the AST structure of the expression of the cached plan was created for and the one of the new computation are compared. If they have the same AST structure, the individual names are compared, and the names, which are not equal are analyzed.

Thereby, the most important factor to consider is the file size. If the file size is smaller than a given threshold, it is not required to recompute the plan.

However, a static threshold is not sufficient for most scenarios, since the ratio of the file size of the not matching NDOs to the matching NDOs matters most. If this ratio is smaller than a specific threshold t , the plan will be reused.

In Figure 6.9 there is only one not matching name. If the ratio of the file size fs_{f2} and the overall file size $fs_{overall}$ is smaller than t , the plan can be reused:

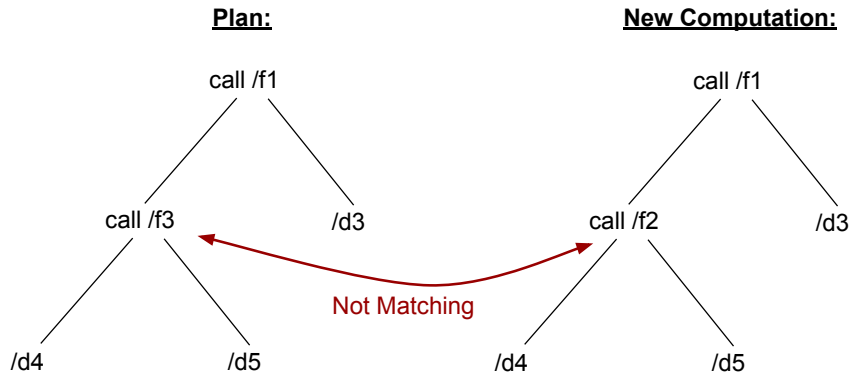


Figure 6.9 Reusing of not exactly matching plans

$$t > \frac{fs_{f2}}{fs_{overall}} \rightarrow \text{reuse.} \quad (6.10)$$

Algorithm 6.5 Analyzing if a plan can be reused.

```

ast_comp = create_ast(comp)
total_filesize = get_total_filesize(plan)
if ast_structure_equal(ast_comp, ast_plan) then
    names = get_names_not_matching(ast_comp, ast_plan)
    for name in names do
        ratio = filesize(name)/total_filesize
        if ratio > t1 then
            return false
        end if
        if filesize > t2 then
            return false
        end if
    end for
end if
return true

```

In Algorithm 6.5 we check for two conditions. First, the ratio of the total size of all NDOs and the size of not matching NDOs is smaller than a threshold t_1 and furthermore, it checks if the size of any not matching NDO is smaller than a threshold t_2 . The second threshold t_2 depends on the speed of the links and can vary. An example for t_1 is shown in Figure 6.10.

The Algorithm 6.5 needs to be run for every named plan available for the planning cache.

This way, the system can prevent, that a plan is requested, if there is a chance that expensive data transferred are issued by non matching NDOs.

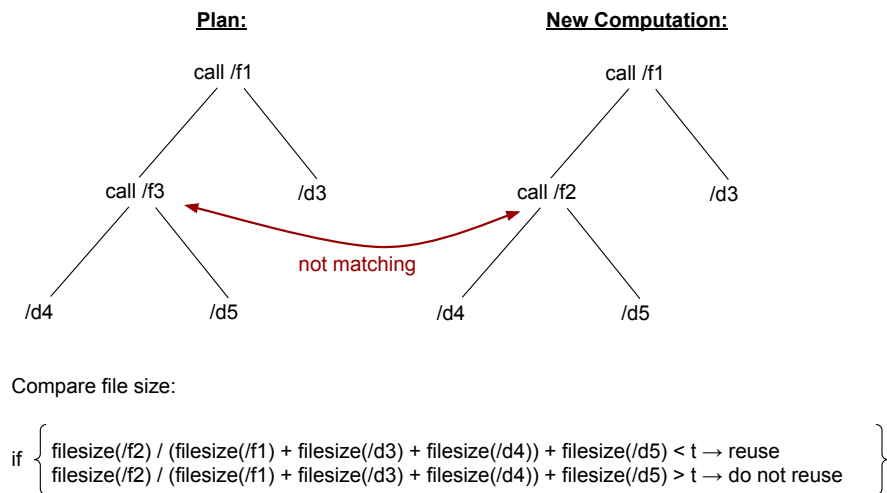


Figure 6.10 Reusing of not exactly matching plans by comparing file sizes

6.5 Presharing of Information Required to Create Plans

Up to this point, we assumed, a NFN node will request all required meta information on demand, i.e. during the planning phase. However, it is also imaginable, that specific information is distributed in a diffuse matter. For example, for information, which is independent of the expression itself, it makes sense to fetch it in advance. This information is for example the load on the network, the load on nodes, the available bandwidth etc.

In the following we will discuss an approach to pre-distribute and to update specific information continuously. Information, which is independent of the expression and the IM is mostly host or link bound. The problem with a host centric approach in ICN is, that there are no host identifiers. Thus, in ICN it is not possible to store the information, that node *a* has load *x*. However, for NFN the execution location is not important, since it relies on prefixes, too. Thus, the important information is, which load is on the node, that holds a specific NDO. In rather static networks with little or no changes on the topology, it is possible to bind the node specific information using the names of NDOs. However, this could lead to a large number of meta information being stored, one entry per NDO, as shown in Figure 6.11

Furthermore, the information about the load on nodes, the load on the network or the available bandwidth is bound to a specific prefix, and is valid, when the prefix is chosen for the local resolution process process, towards a specific face. However, since the next node could change the prepended prefix, result-

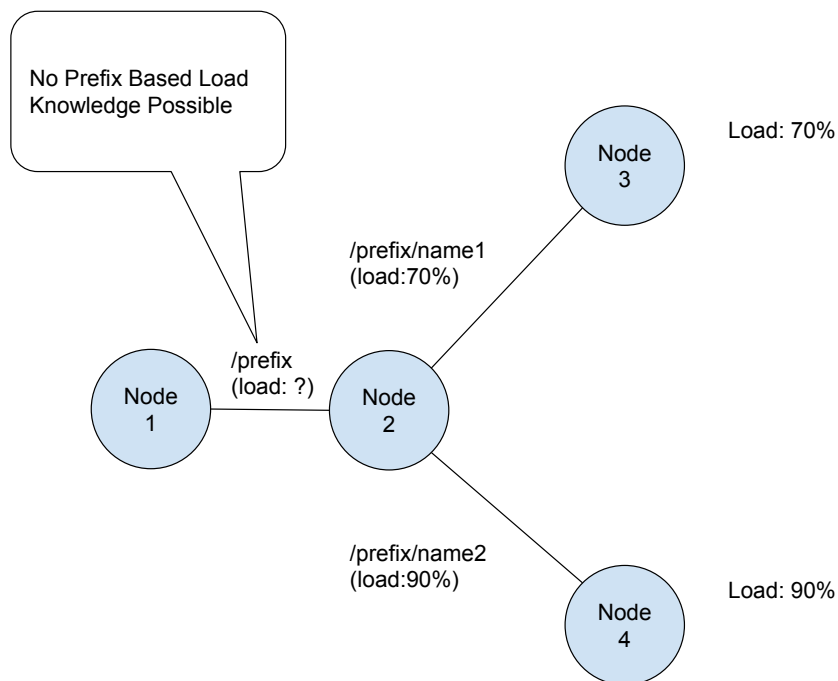


Figure 6.11 Problem with diffuse approach and pre-distribution of data.

Node 3 and Node 4 share the same prefix. Then distributing their load data, Node 2 can handle the case easily, since both prefixes `/prefix/name1` and `/prefix/name2` point to a different FIB entry. Node 1 cannot easily handle the case since it has only a FIB entry for the common prefix `/prefix` and can only distinguish if it stores both specific prefixes `/prefix/name1` and `/prefix/name2`. This would lead to a FIB entry per NDO, which is not feasible for networks with a large number of NDOs.

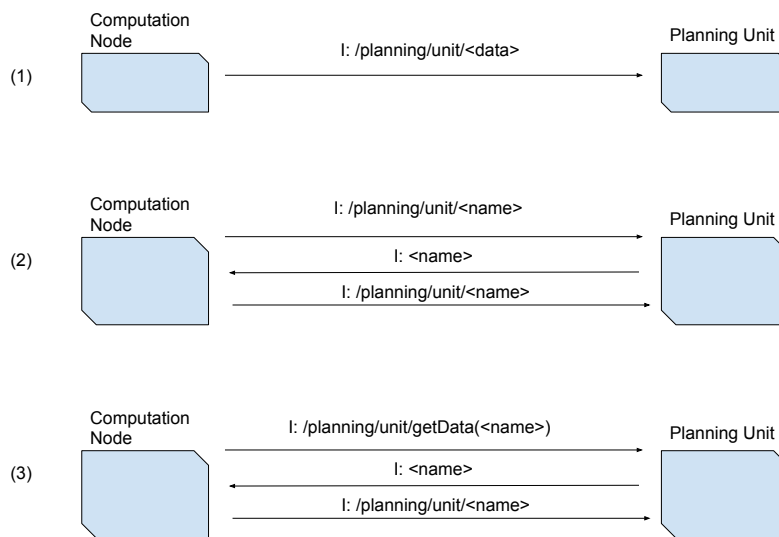
ing in a different execution location, this is not useful. Thus, pre-distributing information is difficult to handle – even in theory – and results in large storage requirements.

Assuming all nodes eligible for the planning process belong to a specific data center or provider, it is possible to use a central entity for planning and information sharing. Thereby, the information about file sizes, load, bandwidth etc, is collected by a central entity (Planning Unit), comparable to a Software Defined Networking (SDN) controller. Since the Planning Unit already knows all meta information, it can directly start computing a plan. In this case, the plan contains all required forwarding information, as shown in Figure 6.2. The *Named Plan* principle (see Figure 6.3) with distributed Plans is not an option here, since the plan is created on a central entity. However, since a central entity can keep plans stored and even update them, the advantages of the *Named Plan* principle can be preserved.

6.5.1 Transporting Data to the Planning Unit

The crucial problem for a Planning Unit is collecting all the required meta information. While the number of nodes – and most likely also the topology – is known, the available data are unknown to the Planning Unit. Therefore, the straight forward solution is, that the computing nodes *push* the meta information to the Planning Unit. However, the underlying CCN network is based on pull-only communication. There are different proposals how a *push* can be implemented in CCN. The simplest solution is to encode data within an IM (Figure 6.12.1). Unfortunately, only a limited amount of data can be stored in an IM, since chunking is only available for content in most CCN implementations. Therefore, another solution is to send an IM with parameters to pull the data from the client (Figure 6.12.2). Therefore, it is required, that the client is addressable by a prefix. Within a data center this requirement can be fulfilled easily. There are specific forwarding rules with a special prefix for the *Planning Unit* and to the computation nodes (e.g. for the *Planning Unit* the prefix could be `/planning/unit`)

The third solution is quite similar to the second one. A pinned function will be stored on the Planning Unit. The purpose of this function is to fetch the data from the computation nodes. Thus, a computation node issues a function call on the Planning Unit, which then requests the data from the computation node, since the *Named Function* automatically resolves the parameters. The purpose of the *Named Function* is just to update the meta information on the *Planning Unit*.

Push Data to the Planning Unit**Figure 6.12 Pushing Data to the Planning Unit**

Three possibilities to push data towards the Planning Unit: (1) encoding data within the IM, (2) sending an IM with the name of the data to be pushed in the name and waiting for the callback and (3) sending a NFN request with the name of the data to be pushed as parameters.

7

NFN Template-Based Resolution Strategy

This chapter describes how to improve the performance of planning in specific cases and how to create *resolution strategies* by planning. For both cases we use templates to generalize plans. Template-Based resolution is about mapping a plan to a class of plans, so it can be used for more than a single computation.

Previously, we reused plans by checking which are locally available. Since this process requires comparing the AST and all names within the AST as well as the sizes of the NDOs.

In the following, we will use the knowledge about plans and reusing plans to create templates.

A template is a plan, where some of the names are replaced by wildcards or at least some of the name components are replaced by wildcards. A wildcard is an empty name or a prefix which is matched against the name of the new computation. If it matches the plan is applied. In this case, it is not required to create the AST anymore, but the action given by the template can be applied directly.

A wildcard can be tagged by a condition to ensure the performance of the template. A condition can be a maximum size of the NDO matching the wildcard or the requirement that the NDO must contain a function. Figure 7.1 shows an example of template matching.

Reusing plans and the creation of templates can speed up computations by pruning the planning time. However, for templates to work it is necessary, that computations are similar and issued within a short period of time as long as the templates are valid. These conditions can be given within a data center, when a service provider offers the same or a similar service to many customers. For

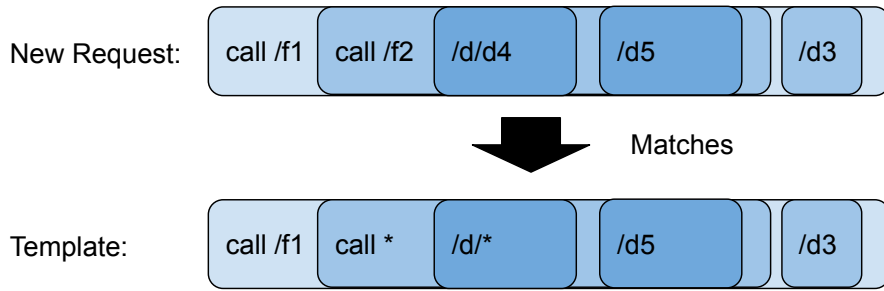


Figure 7.1 Example for Template Matching

The request is matched against a template containing a wildcard.

example if just the input data changes but they are stored in the same region of a data center, templates are expected to work perfectly. However, in scenarios where users define their own computations it is very unlikely that computations match a template. In these scenarios we do not expect big benefits.

7.1 Creating Templates

A template is created by using the understanding that the same action is applied for different but similar requests.

Thereby, we focus on two possibilities how a template can be created:

- Reactive: Understanding of the topology and the prefix matching within the FIB.
- Proactive: Simulation and Planning to create templates based on experience.

While the reactive option focuses on optimizing the planning process by increasing the reusability of plans, the proactive option focuses on replacing a *resolution strategy* with templates to handle specific scenarios such as edge computing. The proactive option focuses on learning from experience instead of creating a scenario specific *resolution strategy*.

Both approaches follow the idea, that n plans can be combined, if they have the same AST structure, there are only a few names differing and the action for all of these plans is the same.

In the following, we will discuss criteria for creating templates and for combining plans.

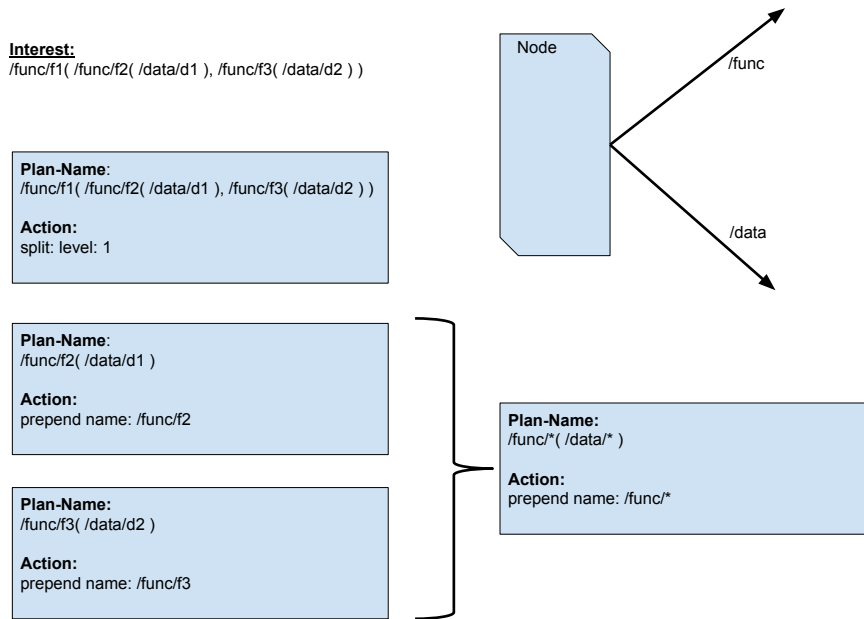


Figure 7.2 Example for Template Creation based on the Topology

Since for both (sub)plans a name is prepended which results in the same action, and there is no other forwarding option, these subplans can be combined in a template.

7.1.1 Templates based on the Knowledge of the Network Topology

The network topology can be used to generalize plans. The idea is to reduce the prefixes of the names within a plan as much as possible, without changing the underlying resolution process. If a plan contains a name consisting of a prefix and a file name such that

`name = prefix / filename`

is the complete CCN name, and there is no better or equal match in the FIB as prefix. The name in the plan can be reduced to the wildcard `prefix/*`, since for all sub(computations) starting with this prefix, there is no other option for forwarding.

Figure 7.2 shows an example for template creation based on the topology. Thereby, for both subcomputations the name, which is prepended, can only be forwarded to a single interface (`/func`). Thus, it is possible to reduce the name in the plan to a wildcard (`/func/*` instead of `/func/f1`). Since the data object `/data/d1` is not prepended, and even if it would be prepended, there is only a single interface to forward a name with the prefix `/data`, we can introduce a

wildcard, too. Since, after introducing the wildcard, both plans are identical, only one is stored.

7.1.2 Templates based on History or Simulation

Templates based on experience focusing on reusing plans are more efficient, while simulation based templates aim to replace the *resolution strategy*.

Both of these forms of template creation are based on the same principle: *Collecting* statistics about the usage of plans and introducing wildcards for popular routes.

The difference here is the time a template is valid. When creating a template from experience, the template is valid as long as the plan is valid. On the other hand, when simulating to create a scenario specific resolution the templates stay valid for a long time or even forever.

A simple approach to template creation is to identify similar plans and check if the applied action for these plans is the same. If it is, they can be combined to a template.

First of all, it is required to check the similarity of the plans. Therefore, we consider every plan with the same structure of the AST. Having multiple plans with the same AST structure is the minimum requirement for creating a template. Next, the names of the AST are checked. If there are different names at the same position of the AST, we check, if the IM is forwarded to the same outgoing interfaces for both plans. The number of plans, which fulfill this property is counted. Thereby, only these plans are counted, for which the different names are at the same position for all plans. Next, the names which are different are checked for a common prefix. Depending on how many components of these names are matching, we create a different score. If the score is high enough, the plans are merged into a template.

The score is computed by the number of identical names and the amount of matching components out of the not matching names:

$$\text{score} = \sum_{n=0}^{n=c_{mc}} mc_n + \sum_{l=0}^{l=c_{nc}} \frac{\sum_{m=0}^{m=c_{ec_l}} ec_m}{e_{ec_l}} \quad (7.1)$$

where mc are the matching names in the AST, c_{mc} is the number of matching names, nc are the non matching names, c_{nc} the number of non matching names, ec_l are the equal components in the name nc_l and c_{ec_l} is the number of equal components in the name nc_l .

Depending on the network topology or the scenario we choose a threshold value t .

If $\text{score} > t$ we introduce wildcards for all particular or not matching names. For particular matching names we only introduce a wildcard for the non matching components.

A wildcard can be connected to a condition. A condition is just an information about the file size or other meta information. For example, if we have only small files for the names which were combined into a wildcard compared to all other files in the request, it is useful to set a condition on the file size, to avoid that the template is used for a computation that might not be a good match. A simple approach to conditions for templates is to prevent applying a template, if the size of the name differs more than factor n from the largest file used to create the template. The factor n can for example be $n = 2$ or $n = 3$ depending on the actual usecase.

In case, there is a plan, which fulfills the criteria to be merged, but the action to be applied is different, we cannot create a template for this plan. Nevertheless, this does not mean we cannot create a template at all. It is just important to define the ordering of the matches.

7.2 Forwarding on Templates

Forwarding on templates basically works the same way as forwarding on plans. If the names match – considering the wildcards – the action of the template is applied.

If there is a template, which matches an expression and there is also a plan, which matches the expression exactly and the action for the template and the plan are different, we always apply the action of the plan. If there are multiple templates or plans matching the expression we choose the template which matches best (There is always only one plan matching an expression, but there can be multiple templates). Therefore, we can use the same score we used to create templates – shown in Equation 7.1 – to decide which template or plan is used. We compute the score for each matching template or plan, and we choose the highest score. If there is a plan, it will be the best match, since obviously the plan has the highest score since it matches exactly.

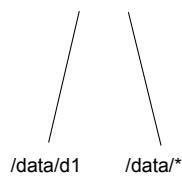
If there is a plan which has an exact match, the score is basically the number of names in the interest message. If there is a template with a best match the score will be smaller. An example is shown in Figure 7.3.

Template and Plan Matching Score

Interest: call /func/f1(/data/d1, /data/d2)

Template 1

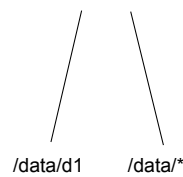
call /func/*



Score:
 $1/2 + 2 + 1/2 = 3$

Template 2

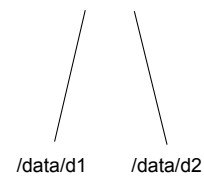
call /func/f1



Score:
 $2 + 2 + 1/2 = 4.5$

Plan

call /func/f1



Score:
 $2 + 2 + 2 = 6$

Figure 7.3 Comparing Templates and Plans using a Matching Score

The score computed with the Formula in Equation 7.1. If multiple templates are matching a request, the best matching template is used. A plan has always the highest score.

PART III

Application Scenarios & Evaluation

8

Application Scenarios for NFN Resolution Strategies

Nowadays, applications are often not executed on a single machine anymore. Parts of the computations are remotely executed or data are not stored locally but produced on demand and fetched from the network. For application developers this means that the complexity during the development of applications increases, since they have to develop separate parts for client and server. This increases not only the development time, but due to the higher complexity, it also increases the frequency of errors and bugs.

Furthermore, if the infrastructure changes, it is often required to change the application code, which is expensive and another potential source of errors.

In Chapter 3 we proposed NFN as a way to separate the application code from the network and suggested concepts how the computations can be distributed over the network. Thus, an application developer can create their applications independently of the underlying network structure and the same application can be executed in different environments without requiring changes to the application logic.

The *resolution strategy* of NFN can be exchanged without changing the application. Stripping away the *resolution strategy* from the actual application logic does not only reduce a source of errors when developing applications, but also reduces the complexity of the application.

In this chapter, we apply the resolution concepts discussed in Chapter 3 to data centers and edge computing scenarios.

First, we start with *resolution strategies* for data centers. Thereby, the focus is on serverless computing, where data center providers offer a service to execute computations. Later, we describe our tests on mobile edge computing with

NFN. Furthermore, we create a network consisting of edge and cloud computing entities and test how different *resolution strategies* work together. At the end we discuss use cases for the *Plan-Based resolution strategies*.

8.1 Applying NFN Resolution to (Serverless) Cloud Computing

Serverless Computing is an easy way to perform computations within a data center. Thereby, an application developer registers a computation and the computation can be executed by a request defining the input parameters. The data center provider chooses an execution location. Computations are stateless functions. Usually, a data center provider chooses a single node for executing a function¹, since it is designed for rather short and easy computations or data transformations. If a function is called multiple times at the same time, it may be executed on different nodes. Current serverless solutions are not designed for more complex computations and function call chaining. While there is no technical limitation, it is more a limitation issued by the business model, which offers serverless computing for easy and short data transformations.

NFN provides exactly this functionality and hooks in where current serverless solutions stop. It provides the same service as serverless computing to let the network chooses where to execute a computation but it in addition offers complex workflow composition and distribution of computations all over the network. A basic use case for NFN in the context of serverless computing is that a data center provider runs a NFN implementation within their data center and offers a service for serverless computing. Thereby, NFN can be hidden from the end user and only be used within the data center in addition NFN can be used to provide the end user the possibility to define and execute their own workflow. This is outlined in Figure 8.1.

Within the data center, NFN can be used to distribute computations over the entire available infrastructure. Thereby, since NFN is based on functional workflow definitions, program execution patterns such as *Map-Reduce* are natively supported.

For the usage in data centers, there are different *resolution strategies* (see Chapter 3) which can be applied:

- *To-Data-First resolution strategy* using only the name of the IM.

¹ see: <https://docs.aws.amazon.com/lambda/latest/dg/scaling.html>

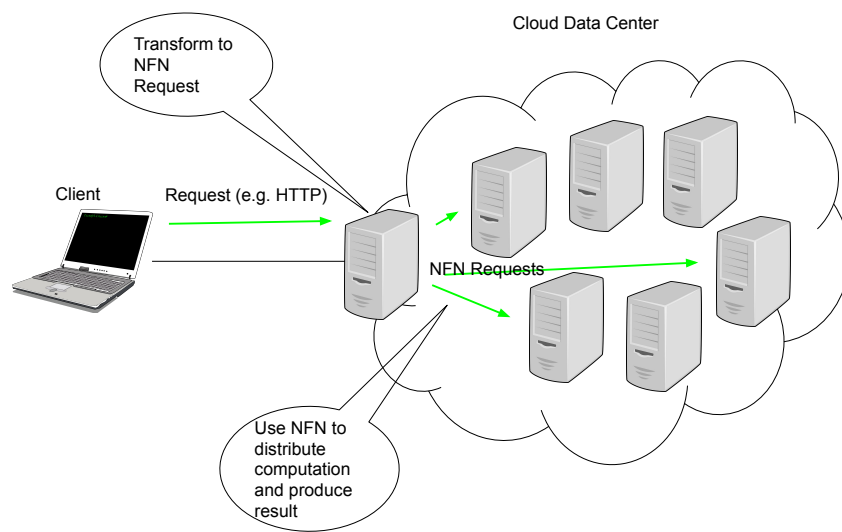
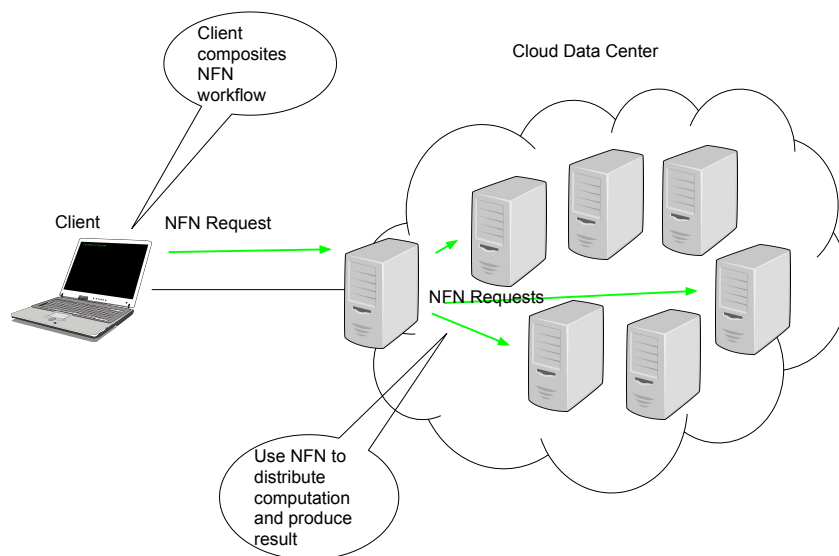
NFN only within Data Center**NFN exposed to the client**

Figure 8.1 NFN for Serverless Computing: Data Center usage only vs exposing NFN to the client.

– *Map-Reduce resolution strategy* also using information from the FIB.

– *Plan-Based resolution strategy* using information about load and data size.

8.1.1 To-Data-First Application

The *To-Data-First resolution strategy* is the simplest possibility in the data center since it simply prepends a name and forwards the IM to this name. It basically executes the computation at the location of (one of) the input data. Thus, NFN behaves like a data center running *Apache Hadoop* but with the option to combine and to compose user defined workflows. This already enhances the options compared to most existing solutions for serverless computing. While *Amazon Lambda* enables workflow chaining for application developers (see AWS Step Functions), NFN additionally enables end users to define their own workflow based on existing or self defined *Named Functions*. Workflow chaining in data center has advantages over computing with traditional remote procedure calls. Instead of downloading data and uploading them again to be processed for every step of the execution chain, using workflow chaining the data and intermediate results can stay within the data center.

The main application for the *To-Data-First resolution strategy* is for *Apache Hadoop* like application, where all input data are stored on the same node of a data center and there is not a lot of variations for different computations. In these cases, the resolution process is as fast as for CCN and there is no additional overhead except on the first node. Since the input data for the computation are already stored close together, it is not required to spread the computation over the network or to perform other optimization steps. Especially, if the input data are stored within the same rack of a data center the communication and the data transfers are very fast, so that it is not required to use more advanced techniques for distributing the computation.

While providing fast forwarding the *To-Data-First resolution strategy* provides a possibility to transport a computation towards a data center or towards a certain location in the data center, it lacks in distributing the computation (see Figure 8.2).

8.1.2 Map-Reduce optimized Application

The *Map-Reduce resolution strategy* starts where the *To-Data-First resolution strategy* is limited - in distributing computations. The basic idea is to split up computations and to distribute them in a way, that each subcomputation is executed close to its input data and if possible on a separate node (within or outside of the same rack). This *resolution strategy* is usable for data centers where the input data for computations are randomly distributed or just even within a rack

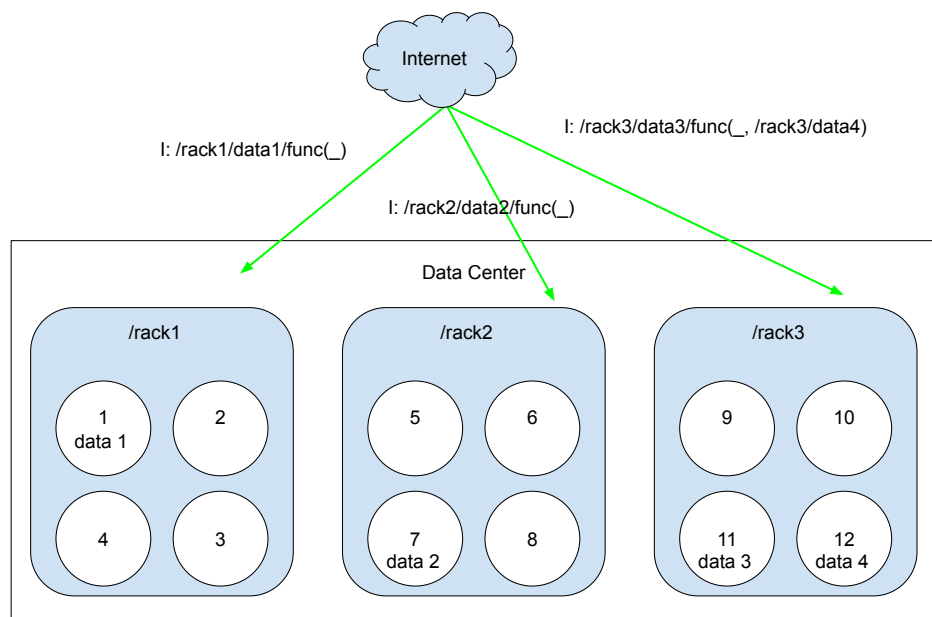


Figure 8.2 The *To-Data-First* resolution strategy within a data center.

As long as all input data are stored within the same Rack, the *To-Data-First* resolution strategy provides a fast resolution process and execution of the computation close to the input data.

to distribute a computation, while the *To-Data-First* resolution strategy was used before to forward the computation towards the rack. This combination of resolution strategies can be useful, since the forwarding decision of the *To-Data-First* resolution strategy are faster, while the *Map-Reduce* resolution strategy is better at distributing computations, which often happens only within a data center or even within a rack. This way, within the “open” Internet, the high speed routing of CCN can be used to forward an IM over rather many hops towards a data center. Later, within the data center, the slower *Map-Reduce* resolution strategy is used over rather few hops to distribute the computation for better performance when executing the subcomputations. An example of this scenario is shown in Figure 8.3.

A second scenario for the *Map-Reduce* resolution strategy is where the required input data for a computation are stored in different data centers. This case, it is helpful, when the *Map-Reduce* resolution strategy is used in the entire network / Internet, so that subcomputations are executed within the data center storing the required input data. An example for this scenario is shown in Figure 8.4.

The distribution of computations to different data centers require that the nodes within the network are also capable of NFN. First, they must implement the resolution strategy for splitting IMs such as the *Map-Reduce* resolution strategy

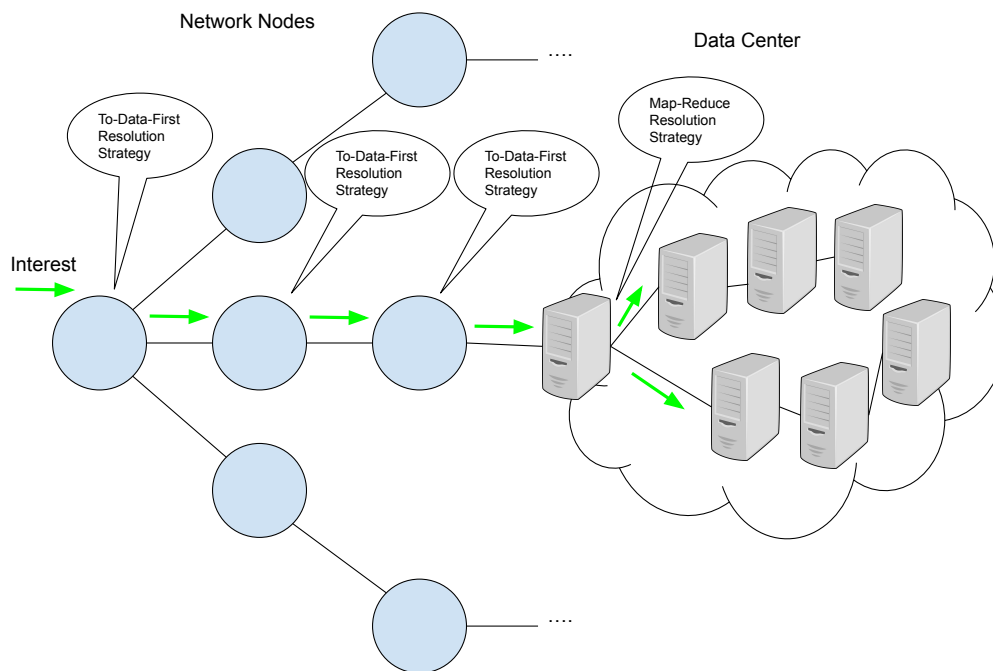


Figure 8.3 The *To-Data-First resolution strategy* outside of the data center to achieve fast forwarding and the *Map-Reduce resolution strategy* within the data center for better data distribution.

and second the node must have the capability to finish the outermost computation for producing the final result out of the subresults produced in the different data centers. This node splitting and forwarding could be placed at the ISP or at the infrastructure provider. However, this may require infrastructure compatible with the one of the cloud provider. In absence of such infrastructure, there are two possibilities for distributing the computation. First, the client splits and combines it by itself, and sends the computation only as subcomputations to the data centers. Second, the client sends the computation to one data center, which splits the computation and sends some subcomputations to another data center. In the case of NFN there is the possibility to use NFN as an abstraction layer for data centers, so that the same application can be executed in different data centers.

8.1.3 NFN and Amazon Lambda/Serverless Computing

In a small experiment we show, that NFN is compatible with existing Cloud and serverless computing solutions.

For example, data can be stored within the Amazon S3 Cloud and NFN can

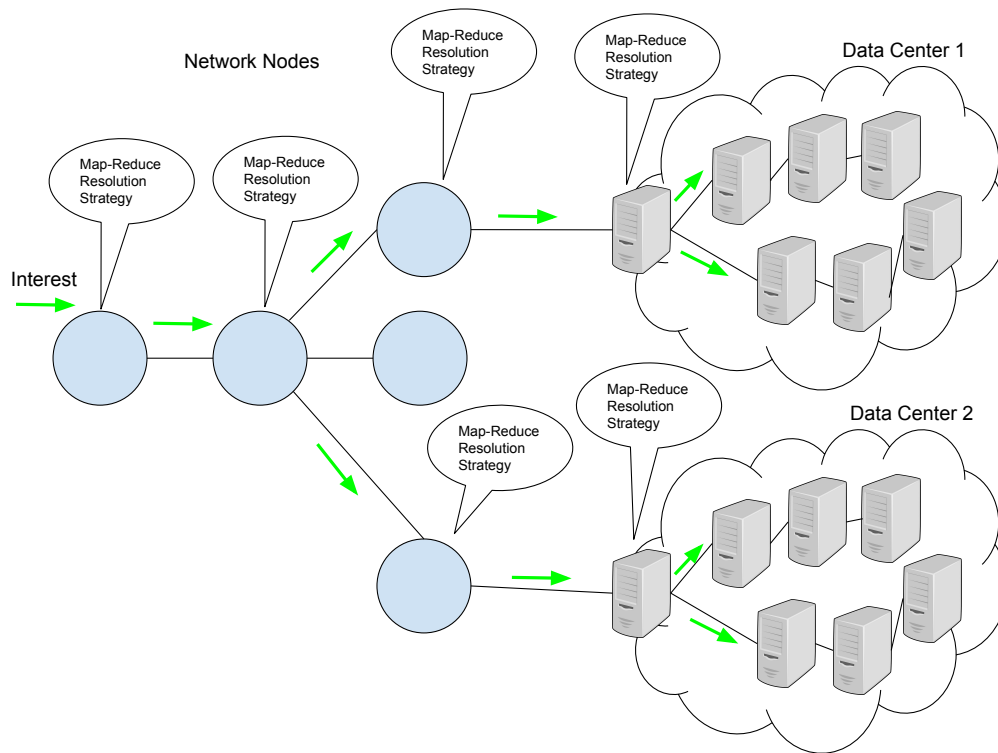


Figure 8.4 The *Map-Reduce resolution strategy* outside of a data center to distribute subcomputations to the data center storing the input data.

be executed on the Amazon EC2 instances. This way, NFN can be an abstraction layer for current cloud computing. Moreover, a named function can point to a Amazon Lambda serverless function as long as the service is stateless. However, to do so, it is required to invoke the serverless function, which requires a node to translate a NFN request to a Amazon Lambda function invocation. Therefore, again, an Amazon EC2 instance could be used. We developed a proof of concept implementation of NFN calling Amazon Lambda Functions. However, this could be done on any cloud, providing Compute and Serverless Instances. This was just performed as a proof of concept and was not evaluated further.

8.2 Applying NFN Resolution to Edge Computing

Beside data centers, edge computing is an upcoming topic. Edge computing moves the computations closer to the client and thereby has several advantages compared to cloud computing. First, the latency between the client and the executing node is reduced. Second, the computations are executed close to the client, thus they are distributed and every edge computing node just serves a few clients. Third, edge computing can enhance the privacy of users, for example the

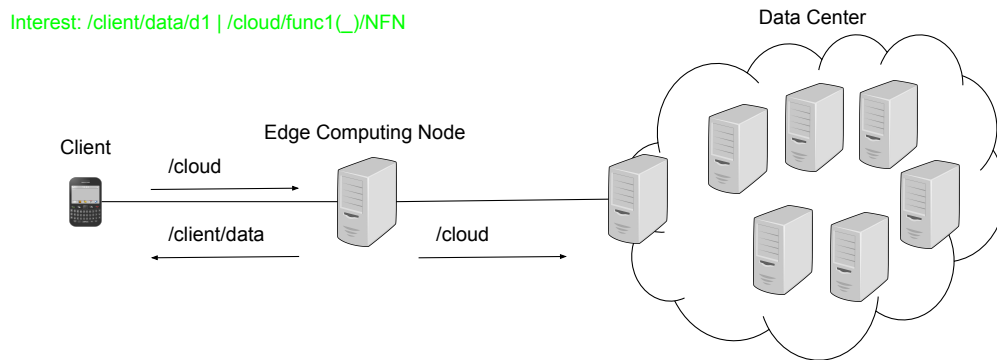


Figure 8.5 Edge Computing Node and NFN

When the IM arrives at the *Edge Computing Node*, it does not forward it back to the client. The *resolution strategy* understands, that the Edge Computing Node should not forward a computation back to the client but execute it locally although all input data are located on the client. The Edge Computing Node needs to fetch all input data.

control system of smart home systems is placed locally, instead of in the cloud. While most smart home systems are cloud dependent, recently some companies offer local controllers ².

For NFN, edge computing means little change of the *resolution strategy* on some of the nodes which are placed at the edge. While *resolution strategies* for NFN are often designed to transport the computation close to the data or to distribute the computation efficiently close to the data, for edge computing it is different. For edge computing the goal is to offload a computation, but forwarding to the data would send the computation back to the client which wants to offload the computation, since it holds the input data or parts of the input data as shown in Figure 8.5.

Thus, the *resolution strategy* on the edge computing node has to analyze the expression and check from where the input data should be requested. If the input data came from the client, it cannot forward the computation back, but either needs to execute it locally or forward the computation further into the cloud. The decision for this can be made by the following rules: If the computation only requires input data from the client it is executed on the edge computing node. If the computation requires data from both, the cloud and the client, by using the planning system, a node can ask which data are larger, the ones from the client or from the cloud and decide by this where to compute. In absence of

² <https://www.bosch-smarthome.com/de/de/produkte/smart-system-solutions/smart-home-controller>, 2018-09-13

the planning system a node can check, whether it can split the computation, so that the part requiring input data from the cloud is forwarded into the cloud and the rest is executed at the edge. The final result will be produced at the edge. If it is not possible to split the computation, the result can be either produced at the edge or in the cloud. Since, we assume data from the cloud are larger, for this case, we produce the result in the cloud.

While we change the *resolution strategy* for the edge computing nodes to apply this decision about either computing locally, splitting or forwarding to the cloud, all other nodes can stay the same.

8.3 Applying NFN Resolution to Mobile Computing

Previously, we discussed static edge computing for NFN. While edge computing is very easy for static scenarios, in scenarios with mobile clients additional challenges arise. In edge computing the nodes are placed – as the name implies – at the edge close to the client. Thus, if the client is mobile, either the distance to the edge computing node is increased or the client needs to connect to a different edge computing node. To maintain the benefits of edge computing a reconnection is required. However, for this case, data upload and delivering results is complicated. NFN provides solutions for this as described in Section 5. We applied this mobile edge computing system for NFN to V2V and V2X scenarios (see Section 2.7).

The focus in this thesis is on V2I communication, where vehicles offload computations to a RSU. Vehicular scenarios are very challenging for edge computing for multiple reasons:

- Vehicles produce a large amount of data, which makes cloud computing complicated.
- Vehicles move with rather high speeds.

To transport a huge amount of data as produced by vehicles over a wireless connection, high frequency bands are required to achieve high data transfer rates. However, on the other hand, high frequency bands have a limited range, which leads in combination to a rather fast movement of the vehicles to many reconnects on edge computing nodes.

For V2I communication, there is one core principle we want to apply: A result of a computation needs to be delivered to the vehicle as fast as possible. Therefore, an Edge Computing Node either computes the result locally or

fetches the result or a partial result from neighboring nodes. Since the node does not know, which way it can produce the result faster, both are tried, and the slower one is aborted as soon as a result is available.

To test the system under real world conditions, we built RSUs and equipped a car with communication hardware. Figure 8.6 shows an image of such a RSU and the car is shown in Figure 8.7.



Figure 8.6 Road Side Unit

A self built RSU based on the IEEE 802.11p WiFi standard.

8.4 Applying NFN Resolution to Multitier Computing

Up to this point, we separately discussed cloud and edge computing. In practice, edge computing usually is not used in separated scenarios but in combination with cloud computing. Thereby, usually edge computing nodes are connected over multiple hops to the cloud. These nodes, for example placed at ISP or at Internet Exchange Points can have additional computational capabilities and form the so-called *Fog Computing Node* area. Thus, Fog Computing is basically a layer between Edge and Cloud Computing, where the definition where a node belongs to is foggy.

For NFN, Fog Computing is another layer, where computations can be executed or can be used to split and distribute computations. For splitting com-



Figure 8.7 Car equipped with IEEE 802.11p communication system.

putations, the same algorithm as for the *Map-Reduce resolution strategy* can be used.

For Edge Computing, the Fog Computing Nodes can be used for efficient data combination, since usually multiple Edge Computing Nodes are connected to a Single Fog Computing Node. Thus, the number of Edge Computing Nodes is higher than the number of Fog Computing Nodes in the network. Moreover, there are less Cloud Systems than Fog Computing Nodes. However, the computational power is highest in the Cloud and lowest on the Edge. This is shown as a scheme in Figure 8.8. Placing computations in a multitier environment containing Edge, Fog and Cloud Computing capabilities, is straight forward. If a computation should be performed, and the input data come from the clients, which are connected to the same Edge Computing Node, the Edge Computing Node will execute the computation (see Figure 8.9). If input data are only accessible via multiple Edge Computing Nodes, but all Edge Computing Nodes are connected to the same Fog Computing Node, the Fog Computing Node will perform the computation (see Figure 8.10). It is possible that there are multiple

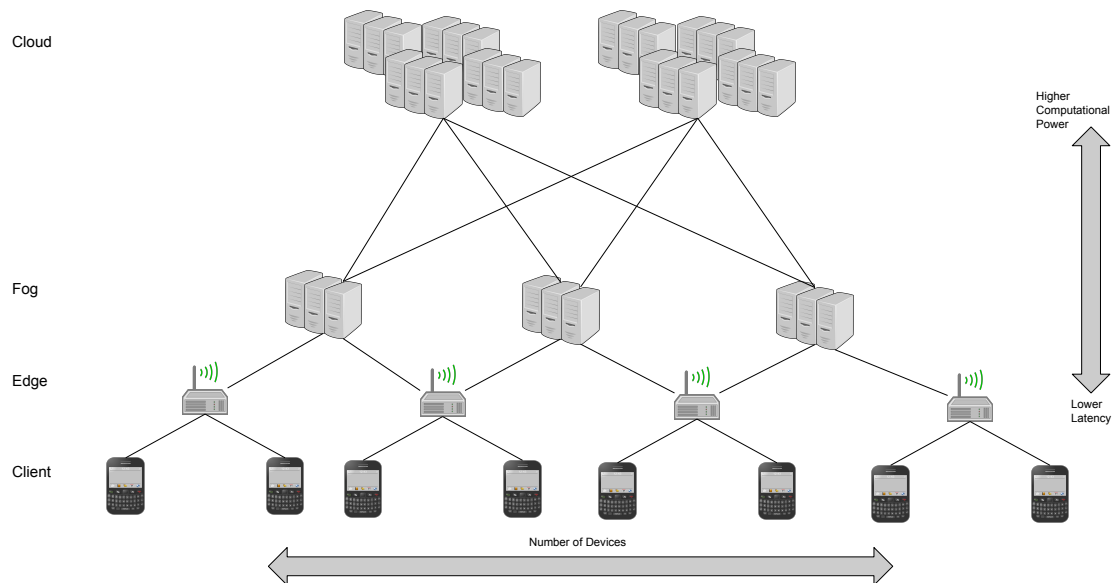


Figure 8.8 Scheme of Edge, Fog and Cloud Computing.

layers of Fog Computing Nodes, some closer to the Edge, some closer to the Cloud. If data are not accessible using the same Fog Computing Node, the computation will be executed in the cloud (see Figure 8.11). However, if it is possible to split a computation into subcomputations, the subcomputations are executed as close to the data as possible. A subcomputation can be moved towards the Cloud and towards the Edge, depending on the input data.

To achieve this behavior we run the Edge-Computing *resolution strategy* on the Edge Computing Nodes, the *Map-Reduce resolution strategy* on the Fog Computing Nodes and either a *Plan-Based resolution strategy* or the *Map-Reduce resolution strategy* in the Cloud. This multitier scenario shows, how NFN flexibly can be used for executing computations in various and heterogeneous scenarios.

In multitier scenarios, multiple *resolution strategies* are used at the same time [ST18]. The *resolution strategy* on the Edge differs for example from the *resolution strategy* in the Cloud. Therefore, it is important to ensure, that these *resolution strategies* are not contradicting each other and sending requests in a loop to each other. A simple approach to avoid unwanted ping-pong behavior is to define the direction in which a computation can be forwarded. A computation can only be forwarded upwards (from client to edge, from edge to fog, from fog to cloud). Only, if the computation is split into subcomputations, the subcomputations can be forwarded either upwards or downwards (towards edge), but once it was forward in one direction, it is not possible to forward it in the other direction. In case, there are connections to reach nodes on the same layer,

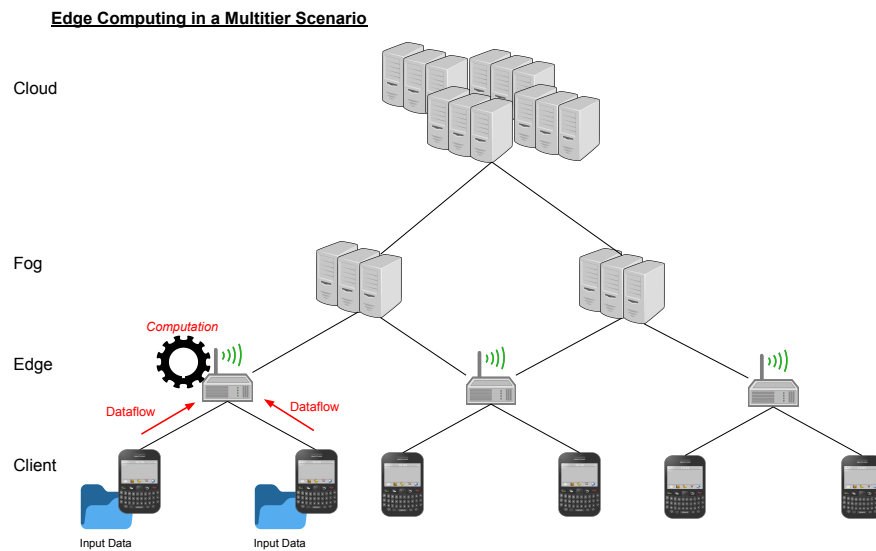


Figure 8.9 Edge Computing in Multitier scenarios.

The input data can be directly delivered to an Edge Computing Node, since both clients holding input data are connected to the same Edge Computing Node. The computation is performed at the edge.

like a sibling Edge Computing Node, a computation or a part of the computation can be moved *side-wise*. This way, we can ensure that first any ping-pong effect is avoided and second, that a computation can be distributed optimally over the network.

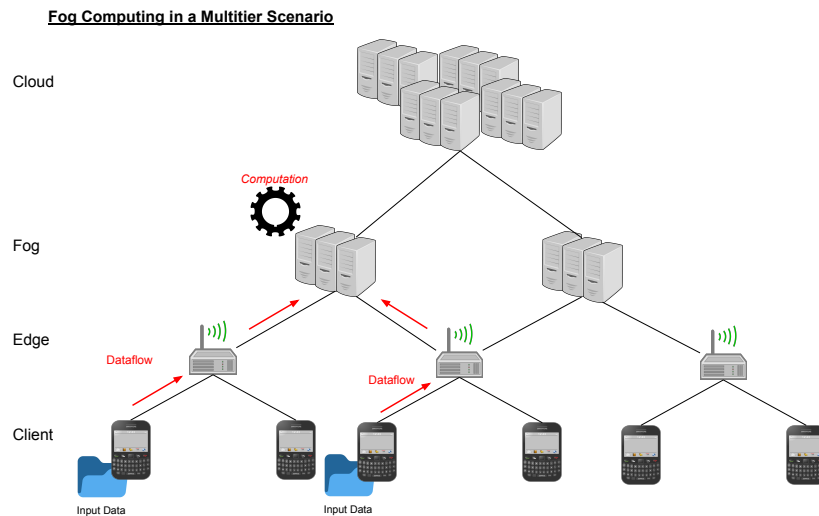


Figure 8.10 Fog Computing in Multitier scenarios.

The input data cannot be directly delivered to an Edge Computing Node, since both clients holding input data are not connected to the same Edge Computing Node. However, since both clients holding input data are connected to the same Fog Computing Node, the computation can be executed on that Fog Computing Node.

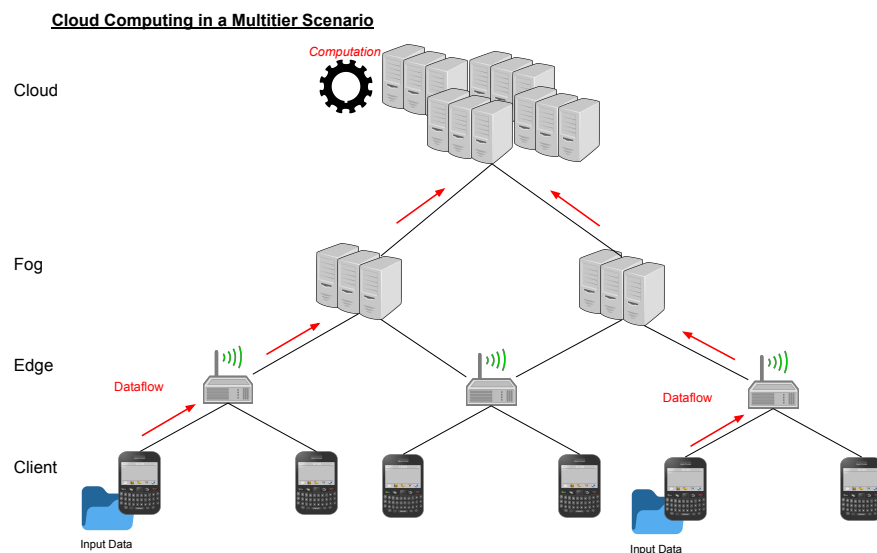


Figure 8.11 Cloud Computing in Multitier scenarios.

The input data cannot be directly delivered to an Edge Computing Node, since both clients holding input data are not connected to the same Edge Computing Node and neither to the same Fog Computing Node. Therefore, the computation is executed in the cloud.

9

Evaluation

This chapter presents our evaluation of the different NFN *resolution strategies*. Thereby, we focus on two scenarios, one for cloud computing, the other for edge and multitier computing. The cloud computing scenario will be part of the multitier scenario.

More specifically, in the scenario for cloud computing we will use simulations to compare our new created *resolution strategies* with the *To-Data-First resolution strategy*. In absence of the possibility to simulate Apache Hadoop the same way as we do with NFN we use the *To-Data-First resolution strategy* as baseline and as state-of-the-art reference. We assume, that the default *To-Data-First resolution strategy* of NFN has the same performance as an Apache Hadoop implementation, since they show exactly the same behavior. Apache Hadoop API offers computing results on the node where the input data are stored and by fetching the function code. However, Apache Hadoop does not offer pinned functions since it is only used within the same data center, while NFN can be used in a wider range.

We measure the performance using different computations in different scenarios.

For the edge computing evaluation we connect the data center evaluation to an edge computing scenario to simulate edge and multitier computing. Thereby, we have several edge computing nodes and mobile clients, e.g. cars at the edge. We evaluate different scenarios – simple computations, computations with intermediate results, data upload, data upload over multiple hops, etc – in simulation. Furthermore, we prove with a real world demo and actual RSUs, that our implementation is feasible.

9.1 Software & Implementation

Our NFN implementation is written in Python and based on the CCN implementation *PiCN*. *PiCN*'s NFN implementation is a modular CCN implementation developed by the University of Basel (during and for this thesis). The idea is that new CCN protocols can easily be deployed and also that repositories and applications for CCN can be developed with a maximum of code reusing. Up to this point *PiCN* consists of over 50000 lines of code. It offers data repositories, forwarders, NFN compute nodes and client tools as well as tests and a simulation system. Following the modular concept of *PiCN* in the NFN implementation the *resolution strategy* can easily be exchanged. Therefore, we use the same implementation for all tests, except for changing the *resolution strategy*. This way we can ensure that all side effects of the measurements are minimized.

PiCN includes a simulation system for evaluation of the functionality of features as well as for performance evaluation. The simulation system enables evaluating the performance of different *resolution strategies*. It supports network connections with adjustable speed and delay as well as executing computations in a single node in serial or on multiple nodes in parallel. This way, the simulations give an idea of the performance in real world.

Beside the *resolution strategies*, the NFN implementation in *PiCN* consists of a sandboxed execution environment for Named Functions written in Python. The sandbox blocks any access to the local machine, so that the forwarding nodes cannot be compromised by malicious Named Functions. During the NFN implementation we also experimented with a native code execution environment. We have a running prototype, but for the evaluation we use Python based Named Functions, since they are platform independent and can be used more universally. Furthermore, it is possible for Named Functions to use native libraries.

9.2 Data Center Evaluation

We use two different scenarios, one with two and one with five different simulated data centers. We use the same topology for each data center since this is very typical and is consistent with the topology used for Apache Hadoop.

The first scenario consists of two smaller data centers, while the second scenario adds three data center with more nodes.

First we take a look at the measurements and later we compare the results.

9.2.1 Test Scenario and Configuration:

In the following we describe the two different test setups for the simulation.

The Test Setup:

- Small data center simulation: two small data center, 100 simulated nodes.
- Big data center simulation: two small data center, three big data center, 400 simulated nodes.

Thereby, a small data center consists of 5 racks with 10 nodes per rack and a management node which connects the racks. Figure 9.1 shows a single small simulated data center.

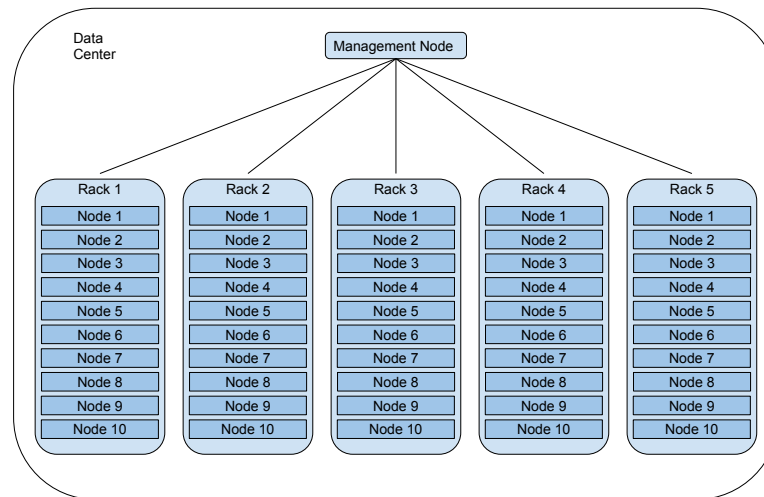


Figure 9.1 Simulated small data center used in the first test scenario.

Data center with 5 racks and 10 nodes per rack.

A big data center consists of two zones. Each zone has 5 racks with 10 nodes per rack and a management node. The management nodes of both zones are connected. Figure 9.3 shows a big simulated data center.

The client is connected over upstream nodes to the data center. A scheme of the small data center scenario is shown in Figure 9.2. There we have only one upstream node, which multiplexes the connection between the client and the two data centers.

For the big data center we use three upstream nodes. Upstream node 1 connects the client with upstream node 2 and 3. Upstream node 2 connects the two small data centers and upstream node 3 the three big data centers. A scheme is shown in Figure 9.4.

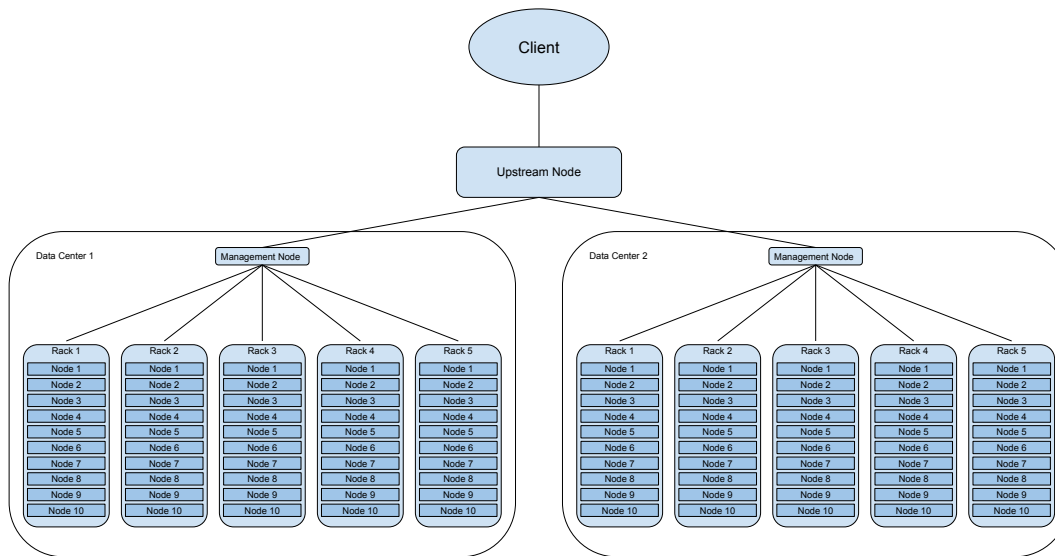


Figure 9.2 Evaluation Scenario with two simulated data centers.

Two small data centers connected by an Upstream Node. The network speed within a rack is 5 times faster than the communication within a single data center and 20 times faster than the communication between the upstream node and the data centers. A node is a computation unit which is connected to a repository which can store data directly on the node.

Network Configuration: Our simulation does not offer bandwidth configured as absolute speed. Therefore, we use a relative configuration i.e. a specific connection is $1/n$ slower than the fastest connection. Our simulated network has different simulation speeds. The fastest connections are between the nodes in a single rack. The connection speed between two racks is $1/5$ of the intra rack connection speed. The connection speed between two zones in the big data centers are $1/10$ of the intra rack connection speed. The connection speed between separate data centers, the upstream nodes and the client are $1/20$ of the intra rack connection speed.

Input Data, Function and Load Configuration: For both scenarios the client issues a computation which requires input data. We distribute the input data randomly over the nodes in the data center. Furthermore, we add a random load to the data centers, whereby the average load is 30% both for the nodes as well as for the network (normal distributed). To make it as realistic as possible we change the load on individual links and nodes during the simulation.

For the measurements, there are three main parts of any execution:

- Forwarding and Distribution Time,

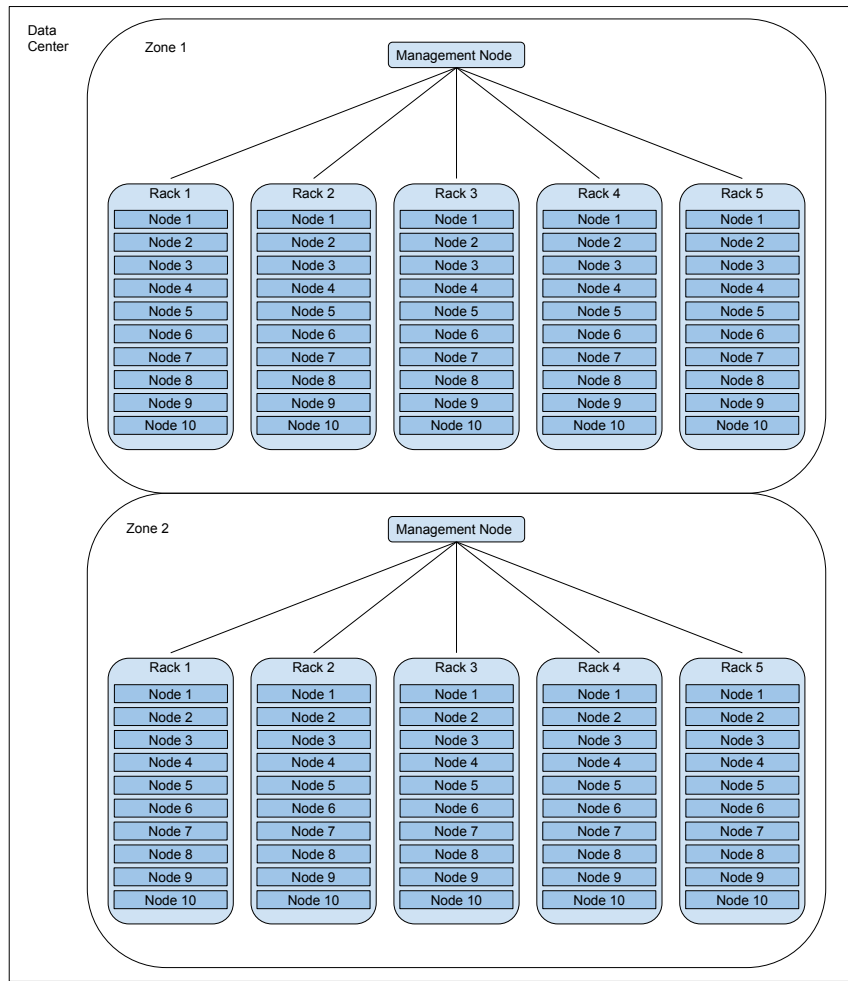


Figure 9.3 Simulated big data center used in the second test scenario.

Data Center with two zones connected over management nodes. The network speed within a rack is 5 times faster than the communication within a zone and 10 times faster than the communication between zones.

- Time for Data Transfers,
- Time for Execution and Delivery of the Result.

To evaluate the functionality of our *resolution strategies*, we measure each of the parts separately.

We evaluate not only a single but six different computations. The difference is, that the input video becomes longer and thus, the computation requires more data transfer time and execution time. Computation 1 has the smallest input video size and it increases till Computation 6 which has the largest input video size. The computation we execute is an object detection in a video stream. It uses TensorFlow, OpenCV and RetinaLib [RDG⁺16]. The input video is split into parts and distributed over the data centers and can be processed in parallel.

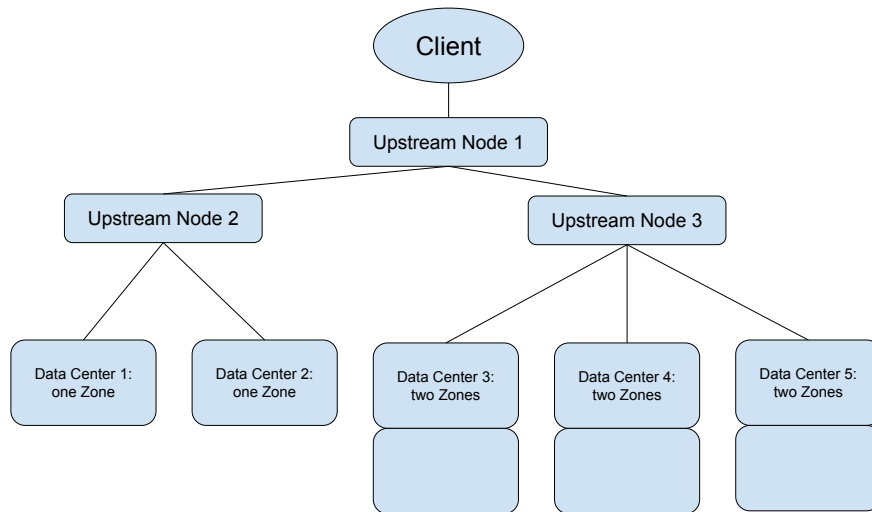


Figure 9.4 Evaluation Scenario with five simulated data centers.

Table 9.1 Number n of subcomputations/input video parts and overall video length

	Comp 1	Comp 2	Comp 3	Comp 4	Comp 5	Comp 6
# Subcomps/Parts	16	32	64	128	256	512
Length Video in s	5	12	28	65	150	400

The parts do not have a unique size, we split the video randomly, while ensuring the smallest part is not less than half the size of the largest part. Our computation itself consists of two parts (outer computation and subcomputations): An object detection on each part of the distributed video and the combination of the results.

Basically, a computation has the following structure:

```
combine( detect (v0), detect(v1), .... detect(vn))
```

for n video parts.

The number n of subcomputations/input video parts and the video length is listed in Table 9.1.

The number of video parts is increased for each computation, thus more parts can be executed in parallel. We also increase their length and thus their data size, which results in more required data transfers and longer execution time.

However, the computation itself is not important for the tests, but only that it requires rather large input data and requires some network communication.

We measure the time in adjusted seconds, which means our simulation is decoupled from the speed of the execution computer, but there is a direct linear dependency between the time to execute in simulation and the real time, so that the simulation results have exactly the same scaling as they would have in a real world test. The time is measured with two-digit precision.

We run each test 1000 times, by randomly reordering the data location and the load on the nodes and the network.

9.2.2 Baseline: Evaluation of the To-Data-First Resolution Strategy

The first evaluation focuses on the *To-Data-First resolution strategy*. This *resolution strategy* was the first NFN *resolution strategy* [SKS⁺14] and it provides the same behavior as Apache Hadoops *compute-close-to-data* API or a similar behavior to the *Task Scheduler* of CFN [KMO⁺19] (CFN offers some more features and thus may have a better performance than the *To-Data-First resolution strategy*, but it is strongly based on the very same concept. Unfortunately, at the time of this evaluation, we did not have access to the CFN implementation nor enough details about the Task Scheduler to emulate it). Therefore, we use the *To-Data-First resolution strategy* as baseline. The advantage of using the NFN implementation as baseline instead of Hadoop or CFN is, that it runs in the same code base as the other evaluation tests and thus side effects are minimized.

When executing our small data center simulation, we end up with the results shown in Figure A.1 (see Appendix).

In the Figure we can see, that the *Forwarding and Distribution Time* is very small compared to the *Time for the Data Transfers* and the *Time for Execution and Delivery of the Result*. The longer the execution time is and the larger the data are (and so the data transfers are more expensive) the smaller is the percentage of the forwarding and distribution time on the overall time while the percentage of the data transfer time increases slightly. This is visualized in Figure B.1. We find, that the percentage of the data distribution time generally increases with larger input files. This is a clear hint, that reducing the required data transfers can increase the overall performance.

In Figure A.2 we see the raw results using the big data center simulation and Figure B.2 shows the percentage of each forwarding, data transfer and delivery time. We can observe a similar result as for the small data center.

Table 9.2 Summary of the Results of the Evaluation of the *To-Data-First* resolution strategy (Mean Value).

The given values are the mean values in seconds for the simulation with 1000 runs.

Scenario	Comp #	Overall Time	Execution Time	Transfer Time	Distribution Time
small data center	1	13.90	8.49	3.90	1.50
	2	48.16	20.80	25.47	1.89
	3	187.30	77.08	107.32	2.90
	4	260.76	104.90	152.95	2.90
	5	385.93	151.22	231.21	3.49
	6	727.06	266.39	455.07	5.60
big data center	1	16.00	8.29	5.90	1.80
	2	60.45	22.30	35.66	2.49
	3	187.30	77.08	107.32	2.90
	4	280.02	103.30	173.21	3.50
	5	396.27	141.23	250.94	4.10
	6	733.10	268.96	458.34	5.80

We see that for the big data center the data transfer time is even larger, since it is more likely data are widely distributed and the subcomputations are executed far away from the input data. Furthermore, the percentage of the overall time required for the data transfers increases faster also for small computations. Thus, for more complex scenarios there is even more potential for improvements.

The idea of all upcoming *resolution strategies* is to invest a little bit more in the forwarding and distribution, but at the same time to dramatically reduce the data transfer time and also to save execution time by more parallelization.

Table 9.2 summarizes the mean values of the evaluation and Table 9.3 the variance.

We see a very similar variance for the distribution time. Since an IM is a small message, the load on the network has little influence on the forwarding and due to the cluster architecture, the forwarding distance is identical for each request.

9.2.3 Evaluation of the Map-Reduce Resolution Strategy

The *Map-Reduce resolution strategy* is the first improved *resolution strategy*. It focuses on better distribution, to compute subcomputations in general closer to their input data and to increase the level of parallelism. We expect to see a

Table 9.3 Summary of the Results of the Evaluation of the *To-Data-First resolution strategy* (Variance).

The given values are the variance for the simulation with 1000 runs. Time in seconds.

Scenario	Comp #	Overall Time	Execution Time	Transfer Time	Distribution Time
small data center	1	0.37	0.24	0.08	0.04
	2	3.67	0.26	3.45	0.04
	3	14.36	2.36	11.33	0.04
	4	42.93	12.97	30.52	0.04
	5	57.59	29.94	26.66	0.04
	6	95.37	60.40	36.77	0.04
big data center	1	0.37	0.24	0.08	0.04
	2	8.21	0.26	8.03	0.04
	3	32.63	2.36	29.22	0.04
	4	95.11	21.44	74.86	0.04
	5	149.25	44.43	102.56	0.04
	6	211.23	96.90	118.31	0.04

shorter time for distributing the computation. Even if the decision time on the router may be a bit longer, the distribution process should be more efficient, since the Map-Reduce *resolution strategy* is capable of splitting the computation on any node and not only if the first executing node is reached. Furthermore, we expect less data transfer time, since the subcomputations will be placed closer to the input data.

Running the small data center simulation with the same settings but only exchanging the *resolution strategy* to *Map-Reduce*, we end up with the results shown in Figure A.3.

The first thing we observe is a slightly faster overall execution time compared to the *To-Data-First resolution strategy* (see Figure A.1). Beside the overall execution time, the time for transferring the data and for executing the computation decreased, while the forwarding and distribution time stayed rather the same.

If we take a look at the percentages shown in Figure B.3, we see that the percentage of the data transfers for some computations even increased. This is because the execution time of the Named Function is relatively even more improved. The better execution time is caused by some parts now being executed on different nodes, which were executed before on the same node.

The most important point which can be spotted in this Figure is, that the percentage of the *Forwarding and Distribution Time* increased when we compare to the *To-Data-First resolution strategy* (see Figure B.1). This is, because it is a slightly higher *Forwarding and Distribution Time* time, but mainly because the other times decreased.

Table 9.4 Summary of the Results of the Evaluation of the *Map-Reduce resolution strategy* (Mean Value).

The given values are the mean values in seconds for the simulation with 1000 runs.

Scenario	Comp #	Overall Time	Execution Time	Transfer Time	Distribution Time
small data center	1	13.50	8.49	3.50	1.50
	2	41.17	18.80	20.48	1.89
	3	149.87	70.08	77.39	2.40
	4	242.63	95.90	143.83	2.90
	5	372.09	147.23	221.27	3.60
	6	695.30	251.39	438.21	5.70
big data center	1	15.10	9.09	4.30	1.70
	2	42.17	19.20	21.08	1.89
	3	154.27	72.38	79.09	2.80
	4	247.96	97.30	147.45	3.20
	5	375.63	147.82	224.01	3.79
	6	702.53	254.98	441.27	6.27

If we take a look at the results in the big data center for the *Map-Reduce resolution strategy*, as shown in Figure A.4, we find that the execution time is closer to the execution time for the small data center, than it was for the *To-Data-First resolution strategy*. We conclude, that the computation distribution for more complex scenarios works more efficiently.

In Figure B.4 we also find, that the percentage of data execution time does not increase as it does for the *To-Data-First resolution strategy*. The execution, data transfer and forwarding time increases slightly with the big data center scenario, we do not observe an overproportional increase of the data transfer time.

Table 9.4 summarizes the mean values of the evaluation and Table 9.5 the variance.

When we take a look at the variance in Table 9.5, we find that all variances are smaller than for the *To-Data-First resolution strategy*. This gives a hint, that the performance benefits of the *Map-Reduce resolution strategy* also come from the fact, that on average it manages to distribute the computation better over the network, no matter how the input data were placed.

Table 9.5 Summary of the Results of the Evaluation of the *Map-Reduce* resolution strategy (Variance).

The given values are the variance for the simulation with 1000 run. Time in seconds.

Scenario	Comp #	Overall Time	Execution Time	Transfer Time	Distribution Time
small data center	1	0.29	0.24	0.01	0.04
	2	2.12	0.26	1.87	0.04
	3	12.99	2.36	10.00	0.04
	4	26.86	11.53	15.69	0.04
	5	45.99	27.80	17.39	0.04
	6	77.88	62.02	17.13	0.04
big data center	1	0.29	0.24	0.01	0.04
	2	2.39	0.26	2.15	0.04
	3	14.76	2.72	11.33	0.04
	4	31.00	12.97	18.44	0.04
	5	45.70	26.76	18.14	0.04
	6	90.46	70.45	21.53	0.04

9.2.4 Evaluation of the Plan-Based Resolution Strategy

While the *To-Data-First* and the *Map-Reduce* resolution strategies only focus to compute close to the data and to distribute subcomputations close to the data, the *Plan-Based* resolution strategy also considers the load on network and nodes as well as data size. For the planning process and the cost estimation, we use the metric as shown in Equation 6.7 and we use for all experiments the distributed planning (we have no evaluation for planning with a Planning Unit). We enable the clustering as described in Algorithm 6.3 for better planning performance. Therefore, we expect less data transfer and faster execution time due to choosing nodes with less load. However, we expect more forwarding time, since the planning process needs more time than the simple interest forwarding.

In Figure A.5 we take a look at the results of the small data center simulation. We notice that the planning time takes much more of the overall time than for the two previous experiments. Moreover, the overall time is faster for Computation 3 to Computation 6 compared to the *Map-Reduce* resolution strategy. With longer execution time and larger input data, the difference rises. When we analyze the percentages in Figure B.5 the planning process takes a lot of the overall execution time for short computations.

For the big data center simulation we see the raw results in Figure A.6.

We see that the overall execution time slightly increases compared to the small data center simulation, but less than for the previous scenarios. One important point here is that the planning time does not increase much, since

Table 9.6 Summary of the Results of the Evaluation of the *Plan-Based resolution strategy* (Mean Value).

The given values are the mean values in seconds for the simulation with 1000 runs.

Scenario	Comp #	Overall Time	Execution Time	Transfer Time	Distribution Time
small data center	1	25.61	7.00	2.10	16.51
	2	45.16	12.80	12.49	19.88
	3	122.95	50.08	47.46	25.41
	4	180.48	65.90	84.65	29.92
	5	310.99	121.23	151.18	38.59
	6	556.44	191.39	318.34	46.71
big data center	1	28.71	7.20	2.20	19.31
	2	47.76	12.90	13.08	21.77
	3	137.64	53.07	50.46	34.11
	4	198.30	67.90	89.27	41.13
	5	336.56	123.23	155.95	57.38
	6	589.19	197.137	323.71	68.11

the planning algorithm is distributed and scales with the increasing number of nodes it runs on in parallel. The percentage of the execution time of each part for the big data center simulation is shown in Figure B.6.

The *Plan-Based resolution strategy* is usable for long computations as typically executed in data centers. However, for short computations it has too much overhead.

The mean measurement data are shown in Table 9.6 and the variance is shown in Table 9.7.

The variance for the Distribution and Forwarding part of the computation increased. We explain this by the fact, that there is a more complex computation involved in the resolution process now and it takes more time, thus it is influenced more by other facts like load or bandwidth. We assume the longer execution time for the big data center simulation is caused by the longer planning time and also by changing load on the nodes.

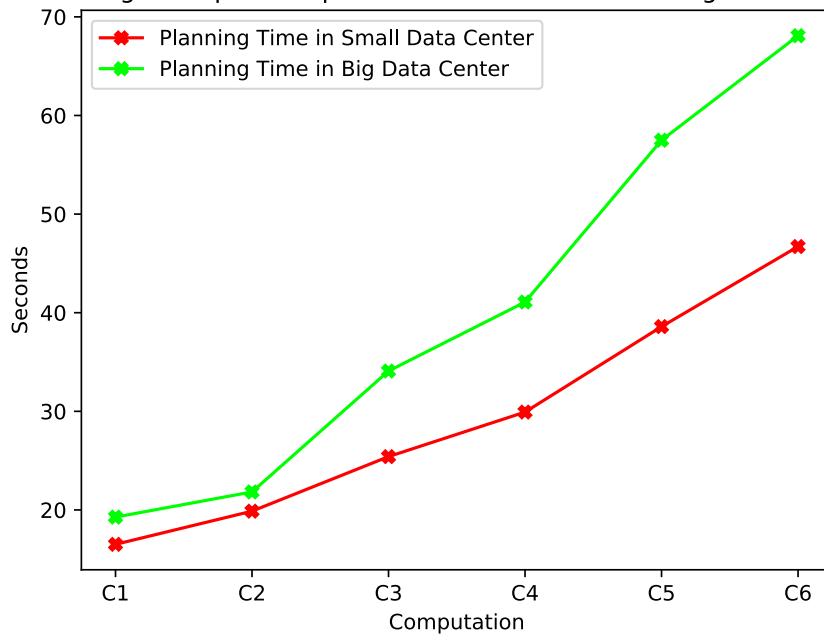
Taking a closer look at the planning time, we find, that the planning process is not increasing exponentially with the number of components and the number of nodes. This has mainly two reasons. In both data center scenarios the clustering works quite well. Furthermore, the algorithm is executed in parallel and scales with the number of nodes. Figure 9.5 visualizes the planning time.

Table 9.7 Summary of the Results of the Evaluation of the *Plan-Based resolution strategy* (Variance).

The given values are the variance for the simulation with 1000 runs. Time in seconds.

Scenario	Comp #	Overall Time	Execution Time	Transfer Time	Distribution Time
small data center	1	0.29	0.24	0.01	0.04
	2	2.12	0.26	1.87	0.04
	3	12.99	2.36	10.00	0.04
	4	26.86	11.53	15.69	0.04
	5	45.99	27.80	17.39	0.04
	6	77.88	62.02	17.13	0.04
big data center	1	0.53	0.24	0.04	0.25
	2	2.42	0.67	1.37	0.49
	3	6.90	3.78	1.76	0.79
	4	18.63	12.97	4.05	1.62
	5	48.89	36.83	7.73	3.23
	6	98.60	88.91	9.19	3.09

Planning Time per Computation for both Small and Big Data Center.

**Figure 9.5 Comparing the Planning Time for both small and big data center simulations.**

Due to the high level of parallelization which is facilitated by our high number of subcomputations, we see even a faster relative planning time for computations with more subcomputations. Figure 9.6 visualizes this by showing the planning time divided by the number of subcomputations.

We also tried to compare the clustering optimized planning with a planning strategy without this optimization. Figure 9.7 shows the results. Due to memory

Planning Time per Computation divided by number of Subcomputations for both Small and Big Data Center.

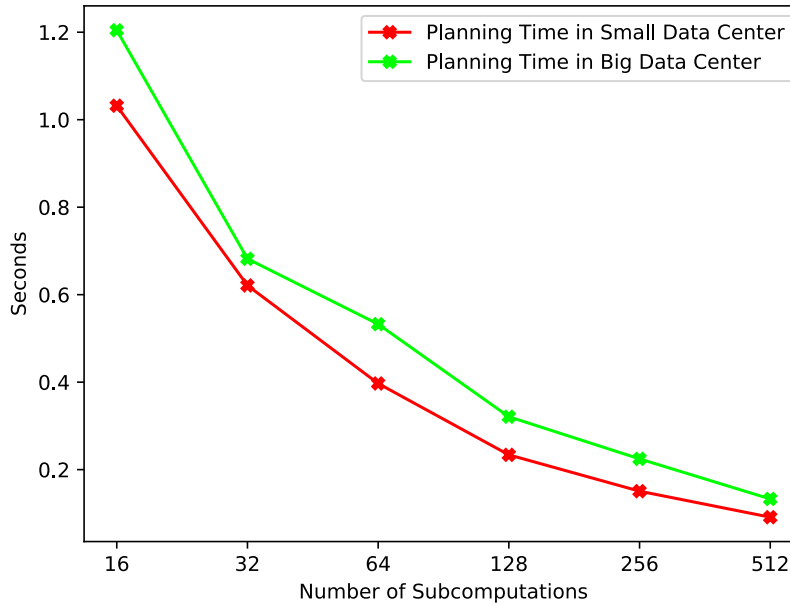


Figure 9.6 Comparing the Planning Time divided by the number of subcomputations for both small and big data center simulations.

Due to the high number of parallel plannable subcomputations, the relative planning time per subcomputation reduces with a higher number of subcomputations.

restrictions without clustering enabled, planning was only possible for the computations c1 and c2. This is why we do not follow up on the planning without clustering and only continue evaluating the planning with clustering. However, the clustering is only efficient for computations with many parallel subcomputations, which can cause problems for the planning with many encapsulated subcomputations in general.

9.2.5 Evaluation of the Template-Based Resolution Strategy

The *Template-Based resolution strategy* tries to create Templates matching more than a single computation during the planning process. Again, we use the metric as shown in Equation 6.7 for the planning process. In this evaluation we first run some computations with similar names to create templates. Later, we evaluate how fast the templates could be applied. The template creation process is fast, it does not increase the time used by the planning process further, but we expect it to dramatically speed up the forwarding when a matching template is available.

The raw results of the small data center simulation with the *Template-Based resolution strategy* is shown in Figure A.7.

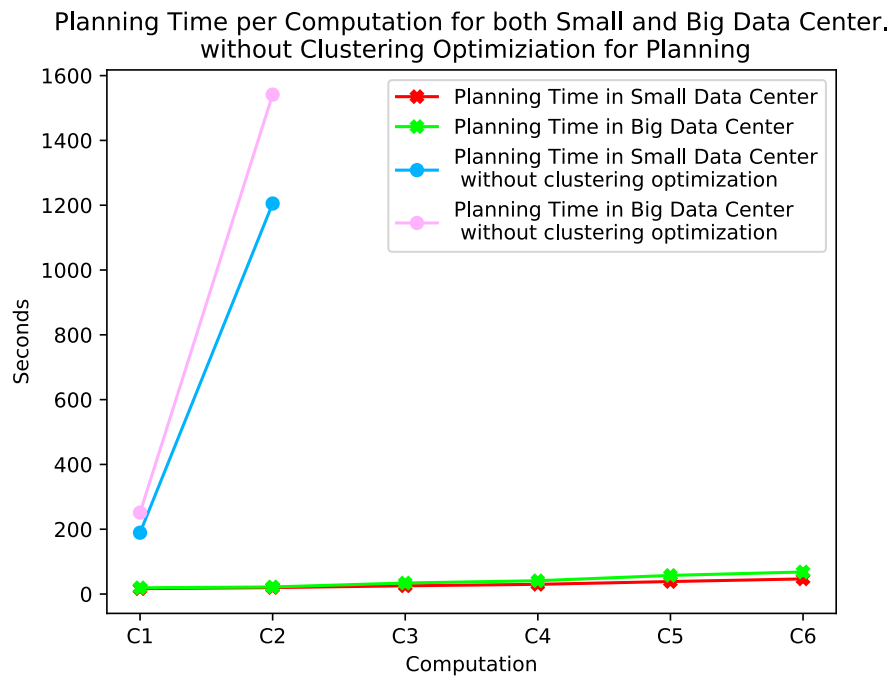


Figure 9.7 Comparing the Planning Time for both small and big data center simulation with and without clustering.

Note: due to memory restrictions, the planning for c3 .. c6 without the clustering optimization failed. c1 had an average planning time of 189.3 s in the small data center simulation and 251.2 s in the big data center simulation. The times for c2 were 1205.2 s and 1541.3 s.

Taking a look at the figure, one can see, that using Templates and execution a computation where Templates are available reduces the overhead during the resolution process dramatically. However, if there are no Templates available, we end up with the same planning times as for the *Plan-Based resolution strategy* (see Figure A.5).

Figure B.7 shows the percentage of the execution time of each part. Again, we see that the percentage of the forwarding and distribution part is reduced dramatically compared to the *Plan-Based resolution strategy* as shown in Figure B.5.

The big data center simulations show the same results with slightly higher overall execution time and slightly higher times in each individual part. We observe, that the usage of templates seems to be less efficient in more complex scenarios, as shown in Figure A.8 and Figure B.8.

The mean measurement data are shown in Table 9.8 and the variance is shown in Table 9.9.

Table 9.8 Summary of the Results of the Evaluation of the *Template-Based resolution strategy* (Mean Value).

The given values are the mean values in seconds for the simulation with 1000 runs.

Scenario	Comp #	Overall Time	Execution Time	Transfer Time	Distribution Time
small data center	1	11.32	7.51	2.10	1.71
	2	29.35	13.98	12.49	2.88
	3	102.64	51.78	47.46	3.40
	4	163.48	68.91	89.65	4.92
	5	291.91	124.14	161.18	6.60
	6	539.96	194.14	338.34	7.47
big data center	1	13.21	8.20	3.20	1.81
	2	33.75	15.19	15.68	2.88
	3	118.99	59.19	56.09	3.70
	4	177.61	74.10	98.29	5.22
	5	319.01	136.14	175.78	7.09
	6	560.79	203.12	347.95	9.72

Table 9.9 Summary of the Results of the Evaluation of the *Template-Based resolution strategy* (Variance).

The given values are the variance for the simulation with 1000 runs. Time in seconds.

Scenario	Comp #	Overall Time	Execution Time	Transfer Time	Distribution Time
small data center	1	0.50	0.24	0.01	0.25
	2	1.23	0.26	0.77	0.25
	3	4.17	2.36	1.26	0.24
	4	15.43	11.53	3.52	0.47
	5	33.22	27.80	4.77	0.16
	6	66.21	62.02	4.90	0.34
big data center	1	0.51	0.24	0.02	0.25
	2	1.97	0.26	1.54	0.25
	3	6.14	3.39	2.07	0.24
	4	21.72	16.10	5.31	0.47
	5	45.33	36.95	7.29	0.36
	6	90.58	83.72	7.85	0.61

In Table 9.8 we see that beside the reduced planning time, for some of the computations the execution and data transfer time increased. This is because the load on the network and on the nodes may have changed slightly. This demonstrates the reason, why cached plans and templates can only be valid for a short time if the metric involves load on network or nodes. Table 9.9 shows a low variance for all the forwarding times again. This is because template matching is very similar to FIB matching and does not involve longer computations such as the planning process.

9.2.5.1 Analyze and Discussion of the Results

In this Section we compare the different results in detail, to get an overview over the performance of the different *resolution strategies*. The goal is to compare the *resolution strategies* depending on the computation which is executed and see which *resolution strategy* performs better when a computation has a certain characteristic.

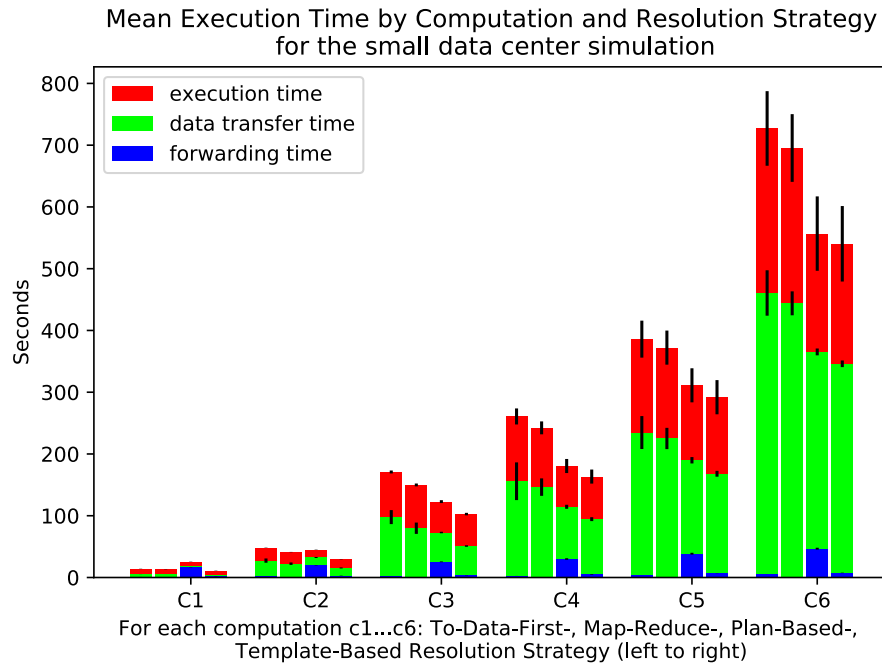


Figure 9.8 Overview over the Simulation Results in the small data center scenario.

The mean and the standard deviation of all computations in the small data center simulation as an overview.

In Figure 9.8 and Figure 9.9 we summarize the results of the small data center simulation and of the big data center simulation. The following conclusions are valid for both simulations.

We first notice, that for all computations the *Map-Reduce resolution strategy* is always faster than the *To-Data-First resolution strategy*. This performance gives us a clear hint to always prefer the *Map-Reduce resolution strategy* over the *To-Data-First resolution strategy*.

For the *Plan-Based* and the *Template-Based resolution strategies* the overall picture is a bit more complicated. We see, that it perform very well by reducing the execution time. However, the forwarding time is increased by the complex planning process. In case, there are matching templates available the planning overhead is removed and the computation is executed faster than with any other

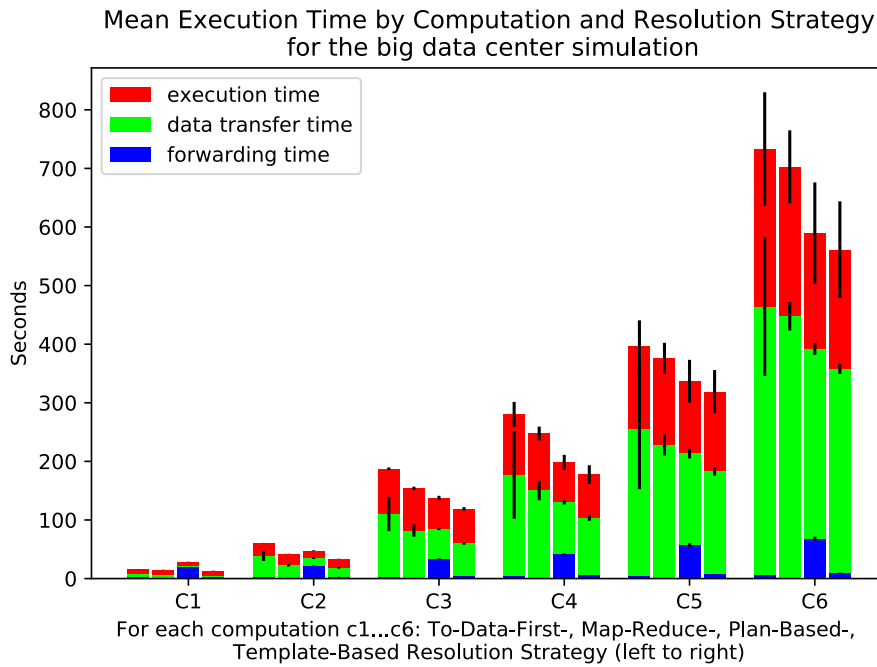


Figure 9.9 Overview over the Simulation Results in the big data center scenario.

The mean and the standard deviation of all computations in the big data center simulation as an overview.

resolution strategy. However, templates are only valid for a very short time period and work better for long and more complex computations, since the chance to find a template for one of the subcomputations increases. Unfortunately, especially short computations are critical for the *Plan-Based resolution strategy*, since the overhead is most hurting for short computations.

A solution for this can be to rate *Named Functions* by their expected execution time or by an execution category. Based on the expected execution time or the execution category the nodes can decide, whether they apply a *Map-Reduce resolution strategy*, or a *Plan-Based resolution strategy*. However, if a matching and recent template is available, it always should be applied due to the performance benefit. This could be explored in future and is not part of this thesis.

We can conclude, that the choice between the *Map-Reduce* and the *Plan-Based* or *Template-Based resolution strategy* depends strongly on the scenario, the requirements and the expected computations. The decision between the *To-Data-First resolution strategy* and the *Map-Reduce resolution strategy* should always end up in choosing the *Map-Reduce resolution strategy* and the decision between the *Plan-Based* and the *Template-Based resolution strategy* should always end up in choosing the *Template-Based resolution strategy*, since for both cases we always found benefits for the *Map-Reduce-* and the *Template-Based resolution strategy*.

9.2.6 Efficiency of Templates

To test the efficiency of our templates we initiate a different test. In the previous tests we explicitly created a Template that will match our target computation, so we can point out the maximum benefit. We define a set of computations $c_1 \dots c_n$ and a set of $d_1 \dots d_m$ input NDOs. We use this sets to generate $p = n/8$ computations with subcomputations. We choose randomly k computations and $k + 4$ input NDOs to construct a computation containing subcomputations with a maximum depth of $k/4$. The computation itself performs dummy calculations just to have a certain execution time. After generating the p computations, we choose $p/2$, $p/4$, $p/8$, $p/16$, $p/32$ and $p/64$ computations to be executed for the creation of templates. Reducing the number of similar computations executed to create templates, the variance increases between the available templates and the later executed computations which should use the templates. This way we simulate a scenario, where a lot of similar computations are executed ($p/2$ computations executed to create templates) and a scenario where a lot of different computations are executed ($p/64$ computations executed to create templates). We execute half of the chosen computations twice, one quarter of the chosen computations four times and one eighth of the chosen computations ten times. This way we try to ensure that some templates are created.

For the execution we use the big data center simulation as shown in Figure 9.4. For our simulation we choose $n = 16384$ and $m = 32768$. NDOs and *Named Functions* are randomly placed in the data center. Each of the six simulation scenarios is run 1000 times. For each run we randomly select new nodes where NDOs and *Named Function* are placed. This way we want to minimize possible side effects for the simulation.

Figure 9.10 shows the results of the experiment.

Thereby, we compare the simulation with the same simulation but with Templates disabled. We find that if less than around 20% of the computations hit a template, the efficiency of the templates compared to planning is close to zero. The higher the percentage of available computation- and subcomputation templates, the faster is the planning time. If templates are available for only a few subcomputations there is a point where the benefit disappears, since due to the distributed manner of the planning algorithm the saved planning process for the subcomputations does not have any impact anymore. However, there might be an impact in the energy consumption, but we cannot evaluate this in our simulations.

Wrapping up, Templates are a very efficient way to save planning time, if the

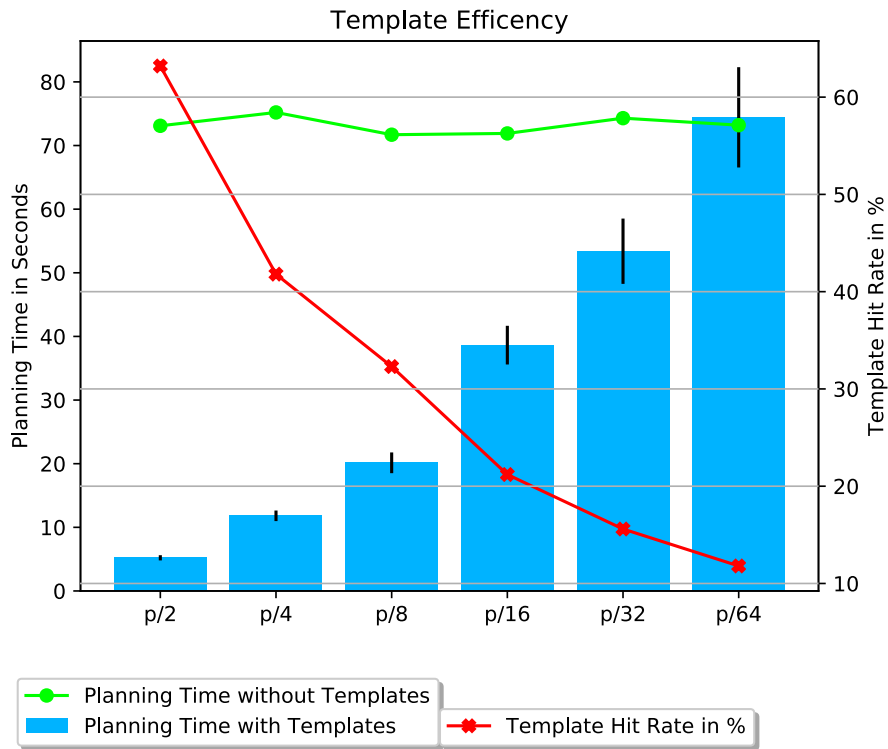


Figure 9.10 Template Efficiency Simulation.

Mean Values of the Simulation Results. In the Simulation similar computations were executed. By reducing the number of available templates, their efficiency drops.

computations are overall very similar. But if templates are available only for one or a few subcomputations, there is no significant effect.

9.2.7 Finding a good Metric for Plan-/Template-Based Resolution Strategies

Up to this point we only looked at a single metric for the execution of the *Plan-Based* and *Template-Based resolution strategies*. In this section we will discuss our metric and our reasons for choosing this metric. Therefore, we will use different other metrics which do not consider specific factors and see how they perform. Thus, our process of finding the metric as in Equation 6.7 was empiric.

We define two different metrics focusing on the load on the node and the data size individually. The focus is on evaluating the extreme cases, where only the load on the nodes and only the data size is considered. We compare this with the strategy we created which is taking both into account equally and in addition the load on the network.

The simulation is run with the same six computations as before in the big

data center scenario.

Figure 9.11 shows the results.

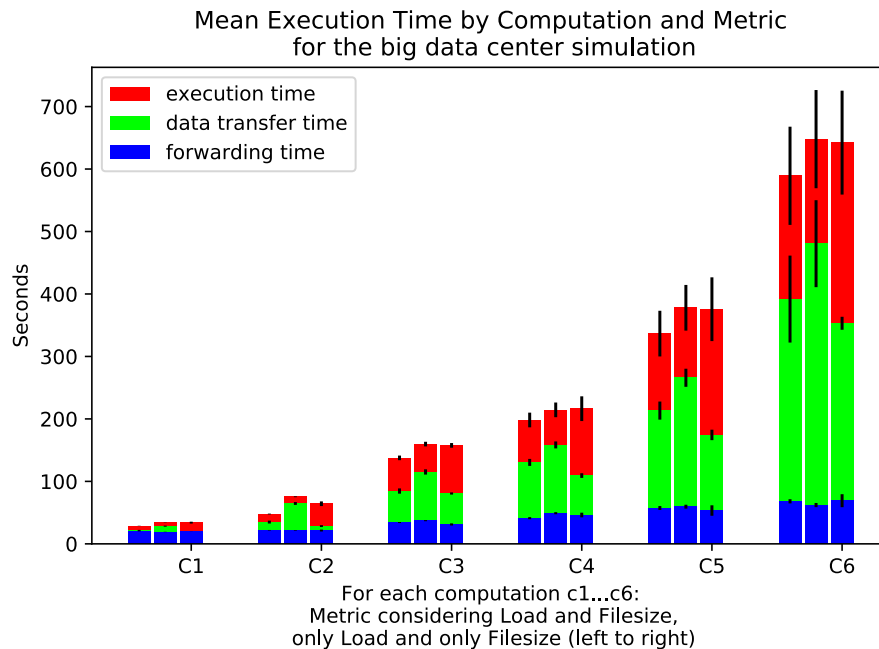


Figure 9.11 Comparing different Metrics.

Comparing different metrics (Datasize + Load on Nodes + Load on Links, only Load on Nodes and only Datasize). We see it is important to consider all factors.

We observe, that by not considering the load on the node the execution time of the named functions increases. As more subcomputations are involved the more the execution time increases overproportionally. The same is true for the datasize. The data transfer times rise even a bit more if we do not consider the size of the data and the available bandwidth. This also proves the concept of computing close to the input data. Another observation here is, that the planning time does not differ a lot. Thus, for short computations the selection of the metric is less critical, since most of the time is consumed by the planning process. We can conclude, that it is important to take all of these factors (Datasize, Load on Links, Load on Nodes) into account. The actually weighting between the individual factors depends on the scenario. For rather fast computations with large input datasize the weighting should be more on the datasize and the load on the links. On the other hand, for more intensive computations the weighting should be more on the load of the nodes.

9.3 Mobile Scenarios Evaluation

In this section we evaluate edge computing with NFN *resolution strategies* for mobile scenarios. For this, we simulate a road, where a car is driving along. The road is simulated by a number of RSUs (NFN forwarders) implementing a *Mobile-Edge-Computing resolution strategy*. Nodes which are simulated cars are handed over from one RSU to the next in a specific time interval. This way, the speed of the car can be simulated. We simulate a scenario where for some RSUs there is a small time between two reconnects, where a car is not connected to any RSU. This simulates the case, where there is a little unconnected area between two RSUs. We simulate different car speeds by increasing the speed of the handover.

The main goal of this simulation is to prove the feasibility of this solution, since we could not identify another work with similar functionality for a baseline. Running computations without an optimization for edge computing will lead to failing to deliver the results at all, as soon as the car reconnects to another RSU.

Later, we present a V2V and a small real world experiment to demonstrate the technique is functional.

9.3.1 Test Scenario and Configuration

In our simulation we use 128 RSUs. Additionally, we use eight cars which are driving along the road. Each 16 RSUs are connected to a fog computing node. The eight fog computing nodes are connected to the five data centers from the big data center simulation.

The connection speed between the cars and the RSUs is $1/5$ of the intra rack connection speed of the data centers. The connection between the RSUs and the data center is slower ($1/20$ of the intra rack connection speed) and shared with others so we simulate a load of on average 30% on it.

The scenario is shown in Figure 9.12.

We perform multiple tests, with two cars and eight cars.

Assuming a RSU has a virtual distance of 500 meters, while the range is around 250 meters in each direction. Thus, if a car drives with 60 km/h it will be connected to a new RSU every 30 seconds, which is simulated by the connection time. In Table 9.10 we list the simulated speed we used for our simulation.

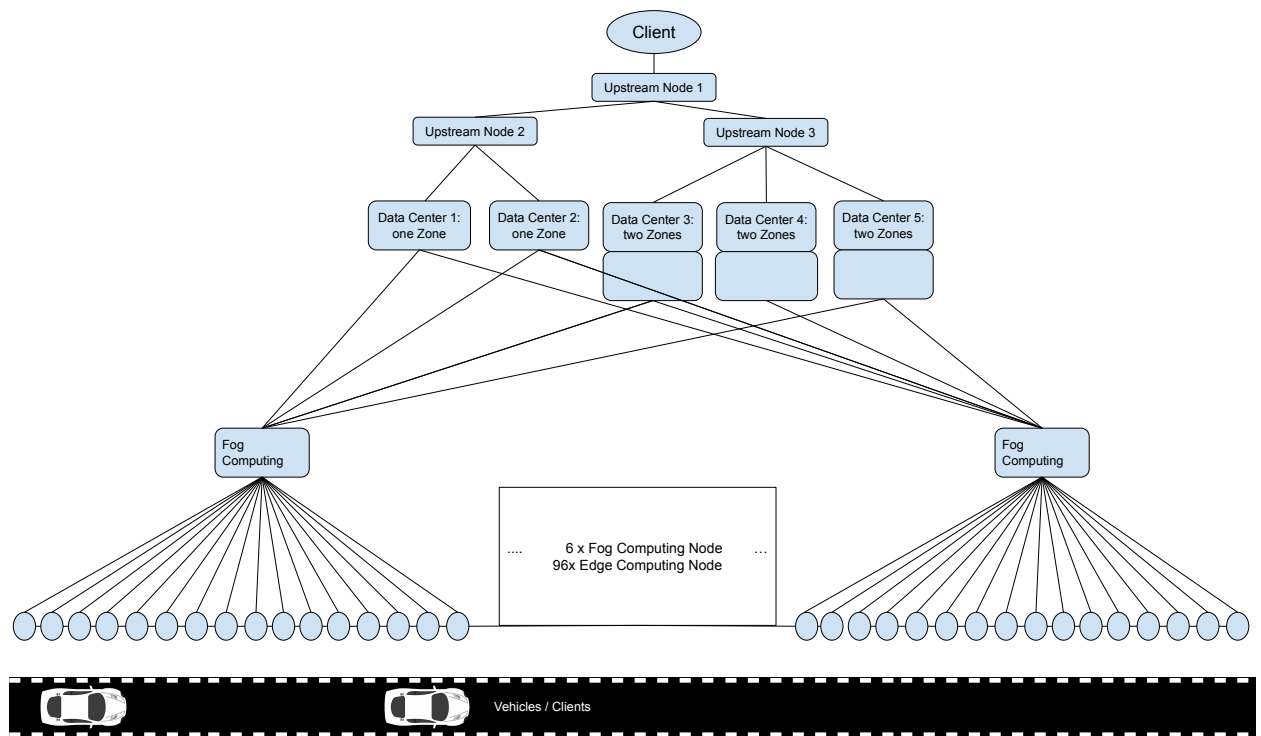


Figure 9.12 Scheme of the testing setup for simulating edge computing.

Table 9.10 Simulated Speed and Connection

	Sim 1	Sim 2	Sim 3	Sim 4	Sim 5	Sim 6
Speed in km/h	60	80	100	120	150	250
Connection Time in s	30	22.5	18	15	12	7.2

For our tests we assume each RSU maintains a register which stores all connected vehicles, so that the vehicles can request which other cars are available. Our test computation consists of extracting features from images and combining them. More concrete, it detects cars, cyclists and pedestrians using RetinaNet [LGG⁺17] and estimates the distance using MonoDepth [GMAF⁺19]. For a better view of the road, we combine the data of all vehicles in range on the RSU. Our example computation is not as optimized as it would be required to be for a real world scenario. The computation takes 10-20 seconds depending on the number of vehicles. Thereby, the data upload makes up roughly half of the time for two vehicles and a quarter of the time for eight vehicles (due to more parallel data upload it is relatively faster for eight vehicles). The cars start their request on a random point in the simulation, thus, they may have already passed 20 of the 128 RSUs. However, this way it is not predictable, if the computation can be satisfied within the connection time to a single RSU.

A computation is for example (depends on connected cars):

Table 9.11 Percentage of delivered results for Mobile Edge Computing with the To-Data-First resolution strategy (Baseline).

Speed in km/h	60	80	100	120	150	250
Delivered Results in % 2 Cars	94.11	86.48	68.62	41.35	19.10	0
Delivered Results in % 8 Cars	51.66	11.52	0	0	0	0

```
/combineResult(/detect(/car/car1), /detect(/car/car3))
```

We place the function code on every RSU and the input data are placed on the vehicles, so it is required to upload them. Note that the result is an object map, which is much smaller than the input data, which are video files.

If multiple cars issue a computation at the same time, it is only executed once. We use a cache time of only one second, since road data are quickly outdated.

We are more interested in proving that the results can be computed and delivered to the mobile client than in the evaluation of the performance.

Each simulation was run 1000 times and the mean value used.

9.3.2 Baseline: Computation with the To-Data-First Resolution Strategy

To show the significance of the problem with edge computing and mobility we first demonstrate what happens when not using a *resolution strategy* optimized for this. The simulation prepends the function code in front of the computation, thus, the RSU will start a local computation. The PIT entry is pointing on to the incoming interface, thus, the result can only be received as long as the car is in range. If the car retransmits the IM when it is connected to the next RSU the same issue occurs.

Figure 9.13 shows the percentage of requests which could be answered. The average computation time with two cars was 10.3 seconds, with eights it was 21.8 seconds. In Table 9.11 one can find the results of our measurement.

We see, that as soon as the connection time becomes short compared to the execution time for the computation, we have a very low percentage of delivered results, since the car reconnects before the result is delivered. However, especially for vehicular scenarios it is critical to have a reliable communication system.

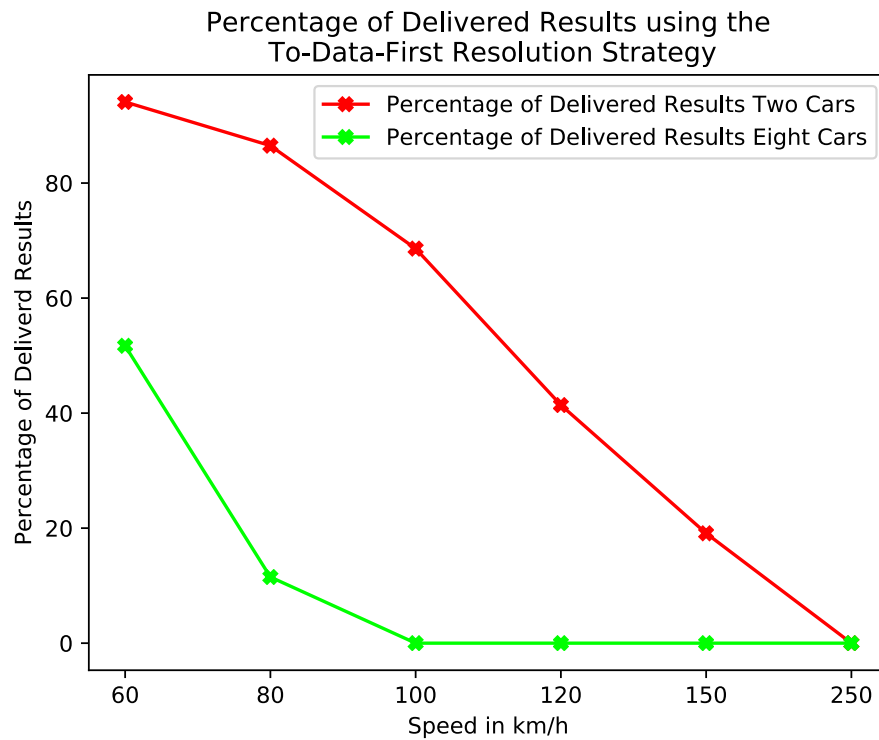


Figure 9.13 Percentage of Delivered Results in a Simulation using the *To-Data-First resolution strategy* with two and eight cars.

9.3.3 Computation with the Resolution Strategy for Mobile Edge Computing

The NFN *resolution strategy* for mobile edge computing is especially designed to handle the case, where the vehicle gets out of range. Moreover, it is designed to deliver what is available faster: cached result or computed result.

Running the same experiment as for the *To-Data-First resolution strategy*, with two and with eight cars along the road, we get a completely different result (see Figure 9.14). In this experiment we have an average computation time of 9.5 seconds for the simulation with two and 19.5 seconds with eight cars. The time benefit comes from some results being delivered from cache.

The percentage of delivered results is now higher than in the case with the *To-Data-First resolution strategy*. However, there is still a high number of computations for which a result could not be delivered. The reason for this is that the data upload takes almost half of the computation time and in case the car connects to a new RSU before the data upload is completed the computation fails.

In Table 9.12 one can find the results of our measurement.

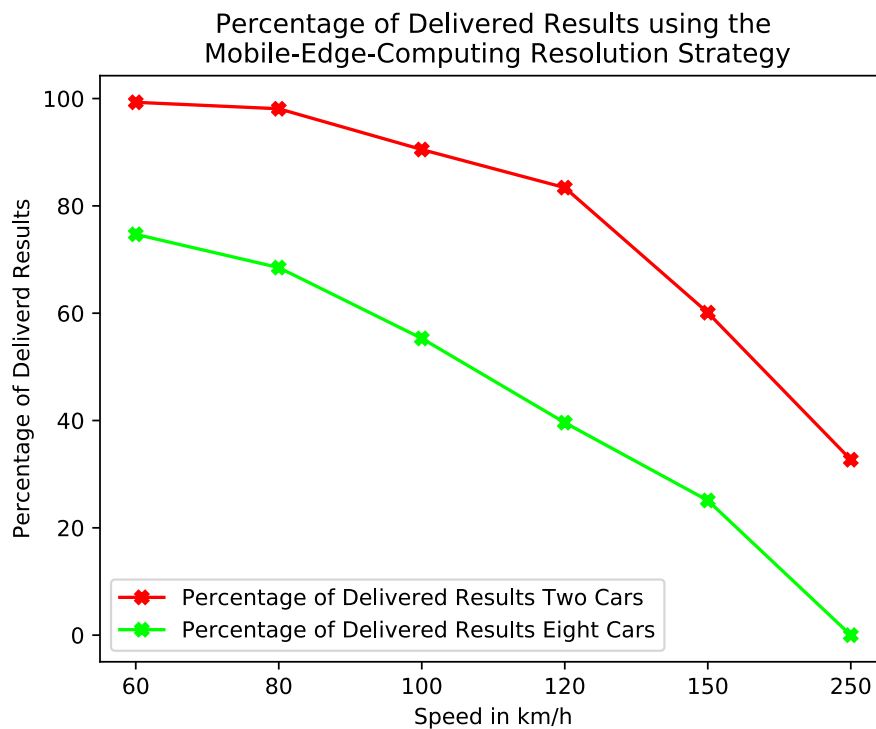


Figure 9.14 Percentage of Delivered Results in a Simulation using the *Mobile-Edge-Computing resolution strategy* with two and eight cars.

Table 9.12 Percentage of delivered results for Mobile Edge Computing with the *Mobile-Edge-Computing resolution strategy*.

Speed in km/h	60	80	100	120	150	250
Delivered Results in % 2 Cars	99.29	98.18	90.50	83.43	60.11	32.68
Delivered Results in % 8 Cars	74.71	68.42	55.23	39.55	25.10	0

9.3.4 Computation with the Resolution Strategy for Mobile Edge Computing and Mobile Data Uploading

In this simulation we repeat the simulation for the *resolution strategy* for mobile Edge Computing. Additionally, we enable the mobile data upload strategy (see Section 5.4). Now, we have an average computation time of 9.3 seconds for the simulation with two and 19.4 seconds with eight cars.

The results are shown in Figure 9.15.

The result is now for all cases 100 %. Thus, we achieve the necessary reliability. All required cases are handled in this scenario, so that a result will be delivered no matter the number or time of the reconnects of the car.

In Table 9.13 one can find the results of our measurement.

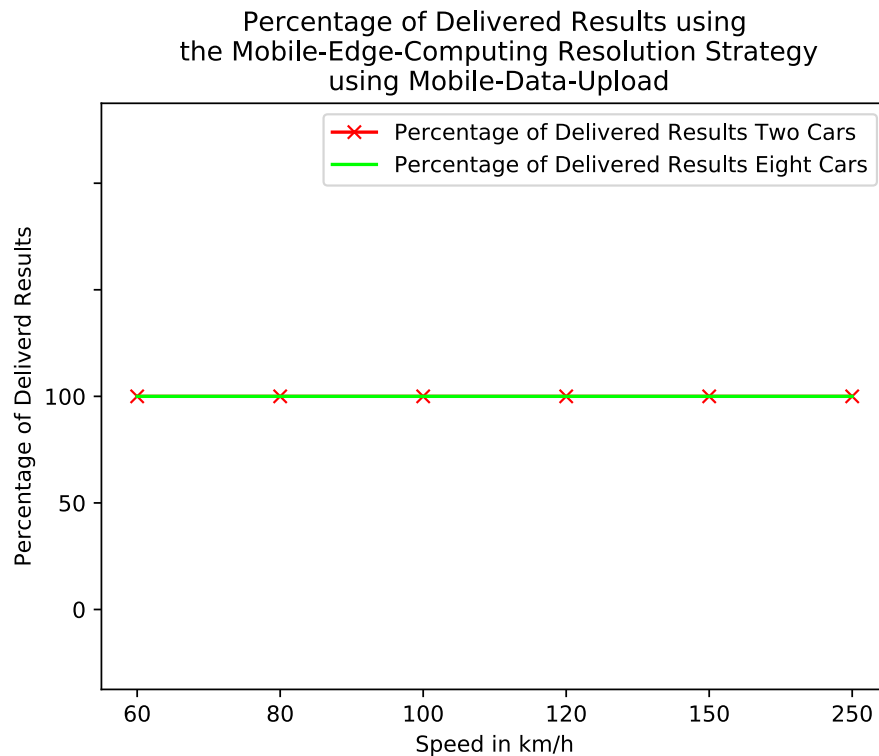


Figure 9.15 Percentage of Delivered Results in a Simulation using the *Mobile-Edge-Computing resolution strategy* and Mobile Data Upload Strategy with two and eight cars.

Table 9.13 Percentage of delivered results for Mobile Edge Computing with the *Mobile-Edge-Computing resolution strategy* including Data Upload Strategy for mobile Scenarios.

Speed in km/h	60	80	100	120	150	250
Delivered Results in % 2 Cars	100	100	100	100	100	100
Delivered Results in % 8 Cars	100	100	100	100	100	100

9.3.5 Comparison Cloud vs Edge Computing

When using cloud instead of edge computing, the mobility scenarios become much easier. By pushing the computation into the cloud (data center) the mobility scenario becomes much simpler, since the cloud is reachable from all edge computing nodes. In this simulation we show the benefits of edge computing for vehicular scenarios. Beside lower latency, the bandwidth between the cars and the RSU is higher than the bandwidth between the RSUs and the data centers. This is a rather realistic property, since upcoming 5G cellular communication for V2X with high frequency offers very high bandwidth.

In the following, we compare the computation time of a simulation where we use the data center and the *Map-Reduce resolution strategy* with the *Mobile-Edge-Computing resolution strategy*. For the data center simulation we place the function code within the data center and prepend the function code in front of the computation. Thus, the interest will be forwarded into a data center and the execution node will pull the input data. We reconfigure the forwarding tables to the cars for each handover to a new RSU so the results can be fetched by the executing node in the data center.

We perform the experiment for two and eight cars and compare the overall runtime and latency to deliver the result after the computation is finished. This latency is important for vehicular scenarios, since for real world scenarios we have very small execution times and the latency is crucial for safety in traffic. The result is shown in Figure 9.16.

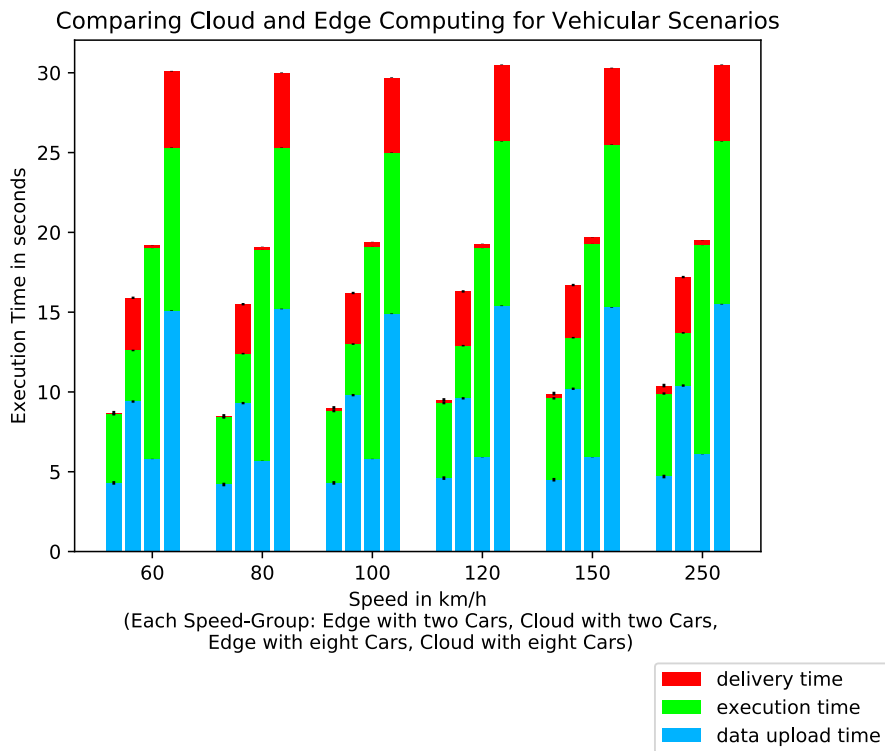


Figure 9.16 Comparing Cloud vs Edge Computing for Mobile Scenarios.

The measurements are summarized in Table 9.14. We find that the faster cloud computing nodes have a smaller computation time, but due to the direct connections the data transfers using Edge Computing are much faster. Thus, for time critical scenarios like vehicular communication Edge Computing seems to be a very good solution. Furthermore, we see, that the overall times do not

Table 9.14 Summary of the Results of the Simulation of the comparison between Cloud and Edge Computing for Vehicular Scenarios. Time in Seconds.

Speed in km/h	60	80	100	120	150	250
Cloud – 2 Cars: Data Upload Time	9.44	9.23	9.81	9.56	10.20	10.39
Cloud – 2 Cars: Computation Time	3.18	3.06	3.21	3.21	3.35	3.38
Cloud – 2 Cars: Delivery Time	3.32	3.09	3.22	3.38	3.32	3.52
Cloud – 8 Cars: Data Upload Time	15.12	15.20	14.94	15.44	15.36	15.57
Cloud – 8 Cars: Computation Time	10.24	10.14	10.03	10.28	10.19	10.25
Cloud – 8 Cars: Delivery Time	4.76	4.68	4.68	4.82	4.81	4.84
Edge – 2 Cars: Data Upload Time	4.32	4.21	4.26	4.61	4.55	4.73
Edge – 2 Cars: Computation Time	4.31	4.16	4.54	4.73	5.07	5.25
Edge – 2 Cars: Delivery Time	0.11	0.12	0.18	0.17	0.22	0.45
Edge – 8 Cars: Data Upload Time	5.82	5.73	5.81	5.85	5.91	6.03
Edge – 8 Cars: Computation Time	13.20	13.23	13.34	13.08	13.35	13.11
Edge – 8 Cars: Delivery Time	4.80	4.73	4.71	4.75	4.86	4.89

strongly depend on the speed. We just find a very small tendency for longer times for the higher speeds.

9.3.6 Including Data from the Cloud and Fog/Multitier Computing

In the following, we look into a Multitier Scenario, adding one data object stored in the data center to the computation, which has roughly the size of 1/2 of the data from the vehicle. We use the Multitier Setting as described in Section 8.4. We create two different test cases:

- We have Fog Computing Nodes available.
- We have no Fog Computing Nodes available.

We run the test with eight cars and we use the following computation:

```
/combineResult(/detect(/car/car1),/detect(/car/car3),/process(/dc1/data1,
/car/car1))
```

The goal of this experiment is to figure out, if Fog Computing can improve the performance. The idea of Fog Computing is here to split the computation and compute the `/detect` part on the Edge and the `/process` part in the cloud or on

the Fog node. Having a Fog Computing Node available gives the possibility not to move the NDO /car/car1 completely towards the cloud and not to move the NDO /dc1/data1 fully towards the edge.

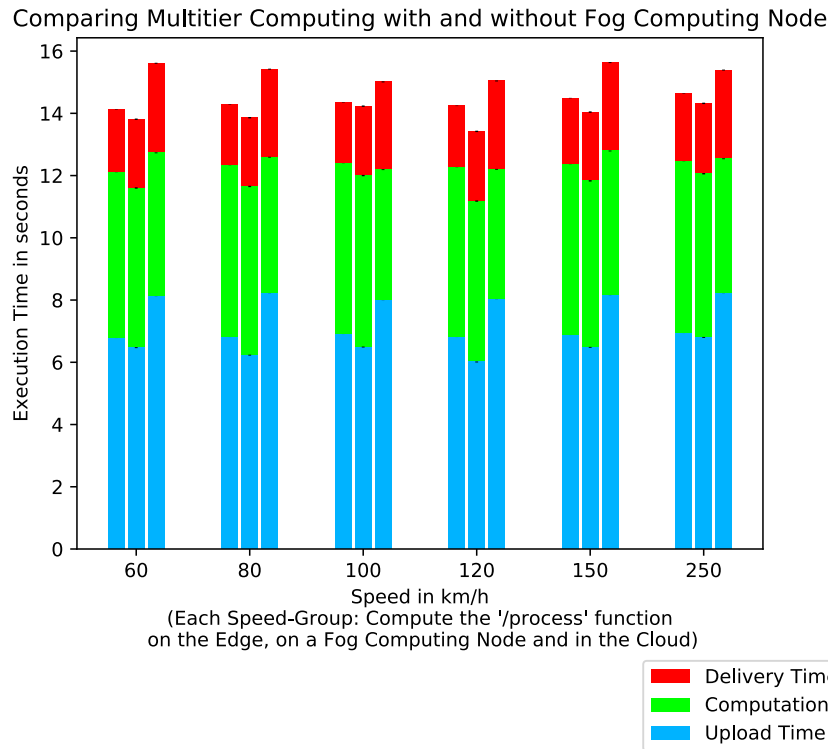


Figure 9.17 Comparing Multitier Computing with computing the /prcoess function on Edge only and in the Cloud only.

The result is shown in Figure 9.17 and in Table 9.15. We do not find a significant difference between computing the part with the Named Function /process in the Cloud, on the Edge or on a Fog Node. Mainly, we observe a bit shorter upload time using the Fog Node, however we do not observe much performance gain by adding the additional infrastructure. Especially, there is almost no difference between processing the /process part on the Edge or on a Fog Computing Node. However, it is possible that Fog Computing Nodes show a bigger performance gain for other scenarios.

9.3.7 Other Experiments

In this section we describe three further experiments serving only as proofs of the concept but no measurements to compare with are available.

Table 9.15 Summary of the Results of the Comparison the/process function only at the Edge, at a Fog Computing Node and only in the Cloud. Time in Seconds.

Speed in km/h	60	80	100	120	150	250
Edge Only: Data Upload Time	6.78	6.81	6.89	6.79	6.85	6.91
Edge Only: Computation Time	5.33	5.51	5.49	5.46	5.52	5.55
Edge Only: Delivery Time	2.01	1.97	1.96	1.99	2.12	2.18
Fog: Data Upload Time	6.47	6.23	6.49	6.01	6.48	6.80
Fog: Computation Time	5.13	5.42	5.51	5.17	5.35	5.26
Fog: Delivery Time	2.21	2.21	2.23	2.24	2.21	2.26
Cloud Only: Data Upload Time	8.12	8.21	7.98	8.02	8.14	8.22
Cloud Only: Computation Time	4.61	4.38	4.21	4.18	4.65	4.32
Cloud Only: Delivery Time	2.88	2.83	2.82	2.84	2.84	2.85

The first is communication in absence of infrastructure (V2V communication), the second is a real world experiment, and third we evaluate templates as basis for *resolution strategies*.

9.3.7.1 Vehicle-to-Vehicle Communication

To show the capabilities of NFN in absence of any infrastructure, in this simulation we remove the RSUs from the road and let the eight cars drive with different speeds (60, 60, 65, 70, 70, 75, 75, 80 km/h). Thereby, each car has own computational capabilities and is seen as an edge computing node. We have the behavior, that the detect functions are executed on the car owning the data, and the requesting car is combining all results together.

We start a computation using the car driving with 70 km/h. When the computation is started, all cars are in range. However, when the results of the sub-computations are delivered, the cars driving 60 and 80 km/h are out of range. We use the mechanism for variable parameters as described in Section 5.6 to be able to compute a result. The simulation proves that the mechanism is functional. A scheme of this simulation is shown in Figure 9.18.

9.3.7.2 Real World Experiment

To prove the concept of NFN Edge Computing in the real world, we built actual RSUs based on the IEEE 802.11p wireless communication standard. A RSU is based on a small computer with a WiFi card supporting the IEEE 802.11p standard. Moreover it runs a NFN relay configured as Edge Computing Node. In this scenario we use two RSUs which are connected by a wired Gbit/s con-

nection. We reduce the transmitter power output of the RSUs so that they have a limited range and it is easier to drive out of range, since the experiment was performed on the area of the Robert Bosch GMBH in Renningen, Germany [GMS⁺18; SGW⁺18]. The car is equipped with the same technology as the RSUs to transmit an IM and to receive the result.

In Figure 9.19 the test setup is shown and Figure 9.20 shows our test vehicle driving along a test RSU.

In this experiment we showed that it is feasible in a real world scenario to transmit a computation to the first RSU and to fetch the result via the second RSU. Thus, this experiment proves the base principle of our mobile Edge Computing concept.

9.3.7.3 Creating a Resolution Strategy using Templates

The idea of creating a *resolution strategy* with templates is to create a *resolution strategy* by simulation (see Section 7.1.2). This can work well in scenarios, where computations are similar and input data having the same prefix. We use one scenario and a small simulation in the Edge Computing scenario to prove this concept. Thereby, we use the *Plan-Based resolution strategy* during the first phase to create templates. Later we apply similar computations. We use the same computation as described in Section 9.3.1. We run first car 1 to 4 to create plans and templates by planning. Later, we inspect the templates and we find that the templates have wildcards instead of the car number. Thus, running the same computation with a different car number (5 to 8) matches the template and the computation can be executed without planning.

For example we run computations like:

```
/combineResult(/detect(/car/car1), /detect(/car/car3))
```

and

```
/combineResult(/detect(/car/car2), /detect(/car/car4))
```

and we end up with a template:

```
/combineResult(/detect(/car/*), /detect(/car/*)).
```

Having this template, we can run the same computation, no matter which identifier is used. Thus, the *Template* based creation of *resolution strategies* seems to be efficient for closed scenarios with similar computations such as our vehicular scenario.

9.3.8 Discussion of the Results

In mobile Edge Computing there is one additional challenge compared to static Edge Computing and to Cloud Computing with mobile clients: Depending on the speed of the mobile client, there is a high probability that the client is reconnected to another Edge Computing Node before the result is delivered. Therefore, we need a mechanism to deliver the result anyway. Our *Mobile Edge Computing resolution strategy* has been proven in simulations and in a real world test. Furthermore, we use a strategy to upload data over multiple Edge Computing Nodes in a way that at the end all data are available on one node to compute a result. Our simulations show, that when not using a NFN *resolution strategy*, which is enabled to handle the mobility, a high number of computed results cannot be delivered. Furthermore, even if a *Mobile-Edge-Computing resolution strategy* is available, there is still a loss due to incomplete data uploads. After adding the data upload strategy, all results can be delivered (our simulations use reliable communication channels, which can be achieved with retransmits in the CCN transport layer, see Section 2.1.3).

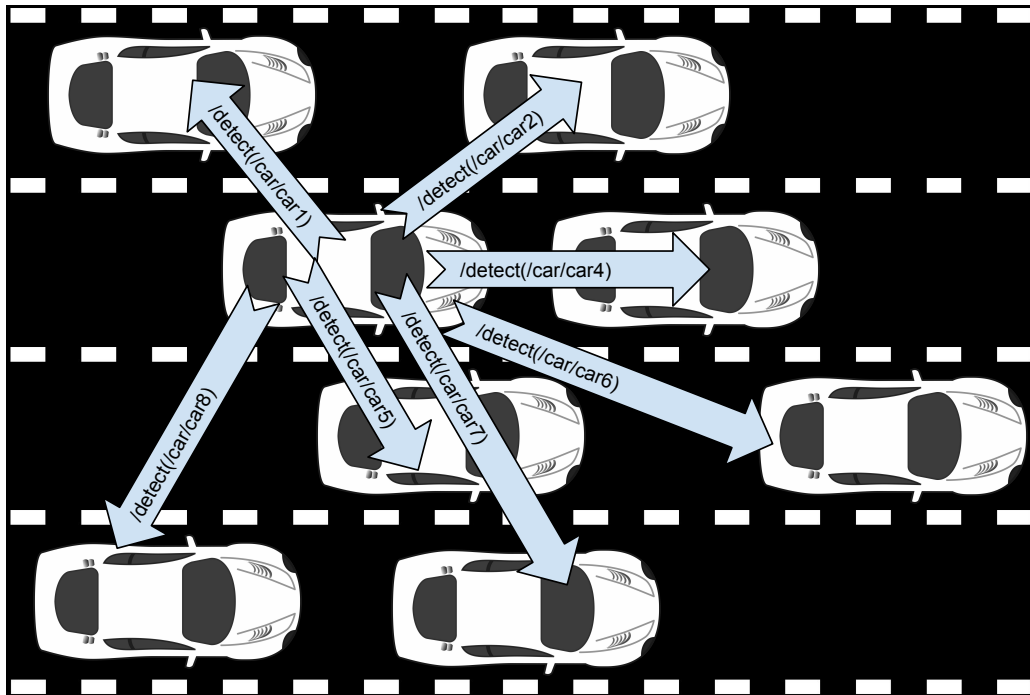
In our simulations we could prove a significant benefit in performance and latency compared to Cloud Computing with Mobile Clients. This benefit should justify the additional infrastructure required for Mobile Edge Computing at least for time critical scenarios such as Vehicular Networking.

Additionally, we ran a simulation to test the performance of Fog Computing, which is a middle layer between Edge and Cloud Computing. We expected to gain performance benefits for scenarios where data from the cloud and the edge are required. However, in simulation we did not find significant benefits that would justify additional infrastructure for Fog Computing. After analyzing our results we find, that in our scenario the Fog Computing offers no benefit, since the file size difference between the data object from the cloud and the result of the computation is not large enough. Thus, the performance difference of Fog Computing and Edge Computing in comparison to the amount of required data transfers is not high enough to gain a benefit. Increasing the input file size and modifying the computation to deliver a smaller result could increase the benefit of Fog Computing. However, in our opinion this is a very constructed setting and thus we conclude that for our vehicular scenario we will not gain big benefits from Fog Computing.

Overall, we ran rather long running computations in our simulations for vehicular scenario. In practice, computations will be shorter, and thus, reconnects to the next RSU will be rarer. However, there is always the possibility that a re-

connect occurs and thus it is required to be able to handle this. Our main reason for testing and simulating with longer computation and data upload times is, that we want to intentionally increase the number of situations where a reconnect occurs, so that we test this situation especially.

(1) Request Inner Results



(2) Collect intermediate Results

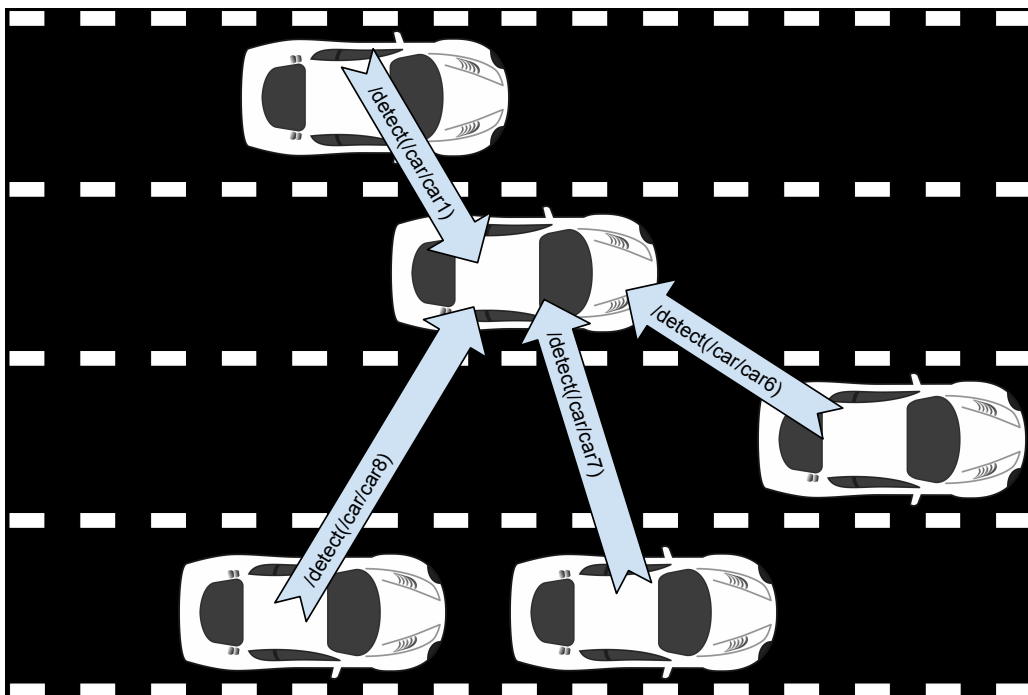


Figure 9.18 Scheme of Computing in absence of Infrastructure.

When the result is collected, not all cars are in range anymore.



Figure 9.19 Test setup for the real world RSU experiment.



Figure 9.20 Test vehicle driving along a RSU.

One can see the wired connection to the next RSU on this picture.

PART IV

Discussion, Conclusion & Future Work

10

Discussion

With NFN we designed a computation framework for CCN. Thereby, we made different decisions for the design of resolution strategies. In the following we will discuss these decisions.

We start by discuss our design decisions and the consequences for privacy. Moreover, we discuss further descision and ideas such as *Billing In NFN*, the *influence of a routing system* on NFN and how we choose a *metric*. Finally, we discuss the results from Chapter 9.

10.1 NFN Design and Privacy

The strength of NFN is that the workflow definition is exposed to the network forwarding and resolution system. This way, the network can understand more about the computation than state-of-the-art frameworks do which use connection based networks instead of a data focused network.

In fact, the whole idea of NFN is based on the fact that the workflow is exposed to the forwarders and thus information about the computations is shared with the network. We utilized exactly this fact for our resolution strategies by analyzing the structure of the computation for the Map-Reduce resolution strategy and even by analyzing the data itself for the Plan and Template-based resolution strategy.

However, exposing the computation to every forwarder in the network could be a privacy concern, since every forwarder gains information about which functions are invoked. Nevertheless, the forwarders only know the previous forwarder and not necessarily the identity of the requesting user. While when used within a data center, where the provider has control over every node and

provides a trusted environment this might not be a problem at all, but used in an open network this could be problematic. If IMs are signed by the sender to ensure authenticity, privacy cannot be guaranteed at all anymore, since the signature always reliably identifies the sender. This could make it very unattractive for a sender to sign its IMs. However, the provider of NFN could block or ignore non signed messages for billing reasons. Billing is a topic we did not consider yet and which should be discussed.

10.2 Billing in NFN

Whenever, one uses a cloud service, they have to pay for the time they used the service. This way, cloud computing has become an attractive business model, particularly for companies that already have large data center available for other business reasons. Especially, the billing models of serverless computing cloud services are interesting for NFN, since the functional function calls in NFN are very similar to the invocation of a serverless function. Usually, serverless function are invoiced per function call. To prevent long running functions, the maximum execution time is capped to e.g. five minutes. Since for NFN the possibility of nested function calls can circumvent restrictions of function calls, in such a billing model, it would be necessary to restrict the number of nested function calls.

However, all these billing models focus on cloud services provided by a single data center provider. One of the idea behind NFN is to act as a *middleware* and to spread computations over multiple data centers of different providers.

For a user it could be now interesting to not only achieve a fast execution, but also an execution using low costs. The classic resolution strategy – To-Data-First – and also in the Map-Reduce resolution strategy forward just on information about the network and the network topology. Therefore, considering prices in advanced can be difficult. There would be options, for example to add a field for expected execution cost to an IM. If a data center node cannot execute the computation for the given cost or less, it will reply with a NACK. An in-network-node receiving the NACK message can then choose another path to compute the result. If no data center node is willing to compute for the given cost, the client will receive the NACK.

On the other hand, the Plan- and Template-based resolution strategy request meta data in the first place to find how to execute the computation at lowest possible cost in regard to a certain metric. In this work we only looked into

metrics which consider technical costs like execution time and data transfer time. However, a metric could easily be extended by the billing cost. This way, a metric can be constructed, which weights between the billing cost and the technical cost. Moreover, a metric could consider a threshold for the billing cost, to ensure, that a computation is not executed to higher cost as the user wants to pay.

An execution model for NFN over multiple data center providers would bring further billing details, since data center providers have different execution mechanism. In general, it is possible to map NFN functions to serverless functions of Amazon Lambda or Microsoft Azure. A second option would be to use the APIs of the cloud provider to start on-demand nodes when needed. This could be part of the planning system.

Caching in NFN in practice would also rise further questions with respect to the billing, for example if a user pays for computing a result and another one gets a cached result for free, who would still compute a result. On the other hand, when paying for cached results, who should earn money: The data center provider, the user or the node which cached the result. To make such a system realistic, the income of caching would need to be split between all of these three entities.

Finally we conclude that taking billing cost in account could be an important, or maybe even the important metric in practical computing, since billing cost often weight higher than performance. The planning system could easily be extended with a metric considering the billing cost.

10.3 Dependency between Routing and Resolution Strategies

The To-Data-First resolution strategy was the first approach to NFN. However, it lacked in possibilities to split the computation on any location inside the network, and thus, it can be suboptimal for distributed and parallelizable computations. The Map-Reduce resolution strategy solves this issue in many points, since it considers not only the availability of the data, but also the forwarding tables of CCN.

At this point, the CCN forwarding system becomes very important, NFN relies on properly filled forwarding tables.

While for CCN there is a lot of work solving routing issues, in NFN we blindly relay on these routing algorithms.

Both, the To-Data-First and the Map-Reduce forwarding strategy assume that the routing algorithm filled the forwarding tables using a metric to minimize the cost. In this work, we assumed a rather optimal behavior of the routing tables and availability of the links. However, it would be also usable to consider the routing algorithm as part of NFN. For example by stabilizing multiple routes and applying load balancing on the routing layer also the NFN resolution system could benefit. While this was not part of this work, the planning system explicitly utilizes multiple routes on the NFN layer. We chose to integrate this load balancing fully into the NFN resolution system, to have the option to apply the knowledge of the NFN planning system and routing together. Nevertheless, we rely on a routing algorithm installing multiple weighted routes and not only the best route, so that we have options to optimize.

10.4 Metrics for the Plan- and Template-based Resolution Strategies

Our Plan-based and our Template-based resolution strategy rely on a metric to rate the execution cost for a certain plan. Therefore, we focused on technical cost, which are relevant for the execution of the computation. We chose the bandwidth, the load on links and the load on nodes as relevant metrics. In our test scenario, where all nodes had the same computation power we could ignore the computational power of individual nodes. However, in practice it can be very useful to consider this as well. It can be retrieved the same way as the computational load, but can also be cached since it does not change.

For our tests, we chose rather complex computations beside the rather large input data and at the same time a metric considering computational power and available bandwidth in a similar way. In case one of computational complexity or data size outweighs the other one, it might be necessary to adopt the metric.

Putting computational complexity in a metric is rather difficult, since it cannot be predicted from the function code itself and can depend on the input data. One way to address this issue is that the developer of a Named Function adds meta information about the expected runtime. Unfortunately, this information can be not precise, since it is manual input. Automatic rating of computational complexity can be done by observation and by caching data of the observations. Measurements about the computational runtime can be used to improve a metric in a network with more heterogeneous nodes.

Assuming we use a way to rate the computational complexity, we need to balance between expected execution time and network transfers. Our current metric does this in a linear way. However, in practice one may want to use non-linear metric, since when ending up with either execution time or data size outweighing the other, the smaller component may become irrelevant and a polynomial or exponential growing metric will lead to better results.

Since the user may know how its computation behaves it could be a good idea to enable the user to define an own weighting of the metric he wants to use. Combining this with billing costs, it could be ensured that the user does wisely choose a metric or they will end up with high billing costs. This factor makes user input here realistic, while a developer of a Named Function does not have additional cost by wrongly rating the execution time of a function and could maliciously manipulate the behavior of the network

10.5 Vehicular Computing and Hand-Overs

Vehicular computing is about autonomous driving, about enhancing safety and about saving energy on the vehicles. Our system is designed to deliver results on different nodes as they were requested and thus to handle hand-overs. It will choose the faster option of receiving the result from the neighbored RSU or computing it locally by applying directly both. Thereby, computations for steering an autonomous vehicles require very low latency and overall computation time. Here a handover of the result from one to another RSU is very unlikely. However, there might be also other operations such as route planning or collecting traffic information or information about the road conditions which can be processed on the RSU to be shared with others. For this kind of computations hand-overs can be beneficial for the performance and for the load in the network. Since especially the transfers of video data generate a lot of network load, it is beneficial for the overall network to execute these computations close to the edge and not to transfer all data in the core of the network. Setting up a new infrastructure along the roads to avoid this kind of data transfers is quite a heavy investment. To cover all roads with RSUs is an effort comparable with setting up a cellular network. Therefore, other solutions like cooperative V2V computations should be addressed, too.

10.6 Analysis of the Results of the Evaluation

In the following we recapitulate the experiments and the performance of our resolution strategies for cloud and edge computing as presented in Chapter 9.

10.6.1 NFN and Cloud Computing

For our evaluation we started with the default NFN *resolution strategy* – the *To-Data-First resolution strategy* which we used as baseline, since it simply forwards an IM as close as possible to the input data and fetches the function code.

Starting from this, we use our new developed more efficient *resolution strategies* for the same computations and compare the performance. With our new defined *resolution strategies* (*Map-Reduce*-, *Plan-Based*-, *Template-Based resolution strategy*) we expect and find mainly performance improvements.

10.6.1.1 Map-Reduce Resolution Strategy

The *Map-Reduce resolution strategy* always performs better. This is mostly, because all our tested computations could be parallelized and thus, the network is better utilized. However, for computations which could not run in parallel, there is very little forwarding overhead. Since NFN can already parallelize a workflow consisting of two subcomputations, almost every computation can be parallelized. Moreover, having less components in the computation (which means it is less likely to be parallelized) the overhead for forwarding is lower. Therefore, we can conclude about that the *Map-Reduce resolution strategy* is always as good or better than the *To-Data-First resolution strategy* and the *Map-Reduce resolution strategy* generally should be chosen over the *To-Data-First resolution strategy*.

10.6.1.2 Plan-Based and Template-Based Resolution Strategy

For the *Plan-Based resolution strategy* this is different. For very long computations we found a large benefit compared to *To-Data-First*- and *Map-Reduce resolution strategies*. However, the overhead of the planning process cannot be ignored. The execution time for short computations increases strongly, the planning process is sometimes longer than the time required for executing the computation. For longer computations the planning process works only if optimizations on the planning process are applied to reduce the complexity. We do this by clustering names. This works well, however, the result of the planning process with our optimization may not be optimal anymore. But our tests show that the results

are still very good compared to the execution and data transfer time of other solutions. Still, there is the problem with the long overall planning time for short computations. We addressed this by reusing plans for similar computations and the creation of templates. For all computations for which a template is available this basically reduces the forwarding and distribution time close to zero. However, this only works, if there are similar computations available. In certain scenarios this can be the case, for example when computations are defined by a provider and users are just using this service. In case of cloud computing there are also scenarios, where users define their own computations, which are then completely unknown and new. In these cases, we do not expect a performance improvement from using templates. For example, Intel (see Section 2.3.5) proposes a concept where steady areas are defined in which data are instantly available [GY19]. This knowledge could be added to the planner of NFN to speed up the planning process. It is a quite similar optimization as the clustering, however, it additionally considers the speed of links, which can be so fast that data are instantly available and is not only oriented on the prefix. By using the planner of NFN, the network has to figure out the nodes for which data are available very fast for each request again, since this information is not cached or reused.

10.6.2 NFN and Mobile Edge Computing

For edge computing, we compared the performance of our *resolution strategy* with the performance of the existing *To-Data-First resolution strategy*. We saw that other *resolution strategies* are not able to handle the mobility of vehicles and results cannot be delivered if the vehicle changes the location and reconnects. Furthermore, we saw that it is required to have a data uploading mechanism available, which is also able to handle the mobility of the vehicle. This way, we achieve a reliable system which can answer all requests from the vehicles. We compared the performance of our vehicular edge computing system with a cloud computing based system. We found that the performance of edge computing for the vehicular scenario is superior due to shorter data transmission paths. Compared to RICE [KHO⁺18], the NFN edge computing solution has the disadvantage that it needs to address the data on the client, thus, the client needs to be added towards the forwarding rules, or a common prefix for all edge nodes needs to be used. However, by not using static bidirectional PIT entries, NFN is capable of handling mobility, even for fast moving clients such as vehicles.

10.6.3 NFN and Multitier Computing

To handle scenarios where input data are required from both Edge and Cloud we try a scenario where Fog Computing nodes are available. However, we did not find big performance differences between using Fog Computing or compute everything on the Edge. Therefore we conclude here that for our scenarios additional infrastructure for Fog Computing does not seem to be worth.

10.6.4 Summary of the results

Overall we can say that the separation of the workflow and the forwarding/distribution process in NFN enables us to execute computations in different environments without changing the definition of the computation. By just changing the *resolution strategy* used by NFN a computation can be adapted to a new scenario. This is very convenient for application developers and for users, since changes to the infrastructure will not break the applications. Performance wise we see a good performance compared to the simulated behavior of existing solutions. We provide NFN and the *resolution strategies* as open source in *PiCN*, so everyone has the possibility to try it and to develop own *resolution strategies* for their scenarios.

11

Conclusion

In this thesis we described a method to extend the network itself to support the forwarding and even execution of computations for cloud as well as for edge computing scenarios. Thereby, we replace the default NFN *resolution strategy* with more advanced *resolution strategies* to consider information about the computation and the network or additional information given from outside such as mobility patterns. We find our results to be promising. In cloud computing scenarios we can observe faster execution times, while in edge computing we achieve high reliability for mobile scenarios.

More concretely, we create a NFN *resolution strategy* utilizing the Map-Reduce pattern for executing computations in parallel. Additionally, we create execution *plans* comparable to plans created for database queries to achieve a faster execution of computations. To create such plans, we request additional meta information about file sizes, load on the network, bandwidth and computational load on the possible executing nodes. Since computing optimal solutions are rather computational expensive, we introduce an optimization method which “clusters” similar CCN prefixes together so that we can consider them as the same. This is possible due to the longest prefix matching properties while forwarding in the underlying CCN. Even though, sometimes we lose the optimal solution, our evaluation shows that with this method we get improved results and without this optimization, the planning would not be feasible at all for computations with more components. Moreover, we came up with a way to reuse plans and to create templates out of frequently used plans. The goal of templates is to match multiple similar requests towards the same plan, so that it can be used for the execution of similar computations instead of only exactly matching ones. This reduces the planning overhead.

Furthermore, we studied Edge Computing which reduces the latency by moving computations closer to the client. If the client holds the input data itself, offloading a computation to an Edge Computing Node is cheaper, since data will not be uploaded into the core of the network. On the other hand, mobile clients may lose the connection to an Edge Computing Node quickly and reconnect to another due to the local manner of edge computing. We developed a concept to upload data and to deliver results after reconnects. This concept is mainly designed for vehicular computing where Edge Computing Nodes along the road are connected to each other. Moreover, we developed a concept for Fog Computing, which places Computing Nodes between the Edge and the Cloud Computing. These nodes should support computations which require input data from user/Edge as well as the cloud.

We implemented our concept and a simulation system to evaluate our Cloud and Data Center Computing *resolution strategies* (Map-Reduce, Plan-Based and Template-Based) in two simulation scenarios. Additionally, we evaluated our *Mobile-Edge-Computing* and *Fog Computing resolution strategies* in simulation. To complete our evaluation, we ran a real world experiment to prove the concept for mobile Edge Computing in practice.

The *Map-Reduce resolution strategy* shows advantages over the default *To-Data-First resolution strategy* of NFN. Moreover, the Plans have proven to be very efficient for long running computations. For short computations the overhead was too much, but Templates can reduce the overhead if enough similar computations are executed. For the mobile scenario, our *Data Upload strategy* and our *Mobile-Edge-Computing resolution strategies* enabled us to reliably fetch results and we could show the benefits of edge computing versus cloud computing for scenarios where the input data come from the client side. However, in our test scenario Fog Computing showed only slight improvements, which leads us to conclude that the additional infrastructure required for Fog Computing is most probably not worth the small benefits.

Overall, we could show that separating computation definition and execution in NFN enabled us to perform computations efficiently in different scenarios without changing the definition of the computation itself. For application developers, the development process becomes more straight forward by removing the complexity communication and network pattern out of the code. Moreover, the results of our evaluation shows that optimized *resolution strategies* speed up the execution and thus releases the developer from certain optimizations.

12

Future Work

In this Chapter we discuss some possible Future Work for NFN and network computing.

12.1 Planning System

Regarding the Planning system, it will be required to improve the performance for short computations, since in these cases the planning time may be longer than the actual execution time. The Templates are already an approach aiming to reduce the planning overhead. However, templates only work if a similar computation already was executed. Here, it is required to find more efficient ways to reduce the planning time. For example, computations could be tagged if they are expected to be very short. For tagged computations, the planning systems could be disabled and the *Map-Reduce resolution strategy* could be used. Another option for improving the planning system would be to use an adaptive metric. For example, if the size of the input data is requested in the beginning and found to be smaller than a threshold, the input datasize and the bandwidth could be ignored and just the load on the nodes be used as the metric. Since a data center could monitor its load, if the load is below a threshold, the planning system could be deactivated and the *Map-Reduce resolution strategy* could kick in.

12.2 Edge Computing

For the *Mobile-Edge-Computing resolution strategy*, we only looked into reactive mobility patterns. The moment a vehicle reconnects to another Edge Computing Node the network reacts and changes the delivery path. However, if a vehicle

would transmit its upcoming behavior, it may be possible to compute a result on the location, where the vehicle will be, when the result is available. This way the latency could be further reduced. For example, for a vehicle on a motorway, the speed is rather constant and the location where the vehicle will be in 2 seconds or in 5 seconds or even in a few minutes is easily predictable. In urban scenarios, it is harder to predict the exact location where a vehicle is heading to since there are pedestrians, cyclists, crosswalks, etc which may interfere with the prediction. Therefore, such a “*Behavior Based*” resolution strategy may be good for motorways but less suited for urban regions. However, to support this hypothesis some research and experiments would be required.

In our experiments we assumed that we know which vehicles are connected to an Edge Computing Node, so we could address the data which were available on these vehicles. However, in reality, it is required to have a directory service, which stores the available data. Therefore, a pinned function could be installed on each node which is called when a vehicle connects and has a sliding window using a ping to detect when a vehicle disconnects. The function stores the data available on that vehicle in a log. This would be a special non-side-effect-free function in NFN. For non-side-effect-free function or functions maintaining status in NFN, it is required to do more research. Using such a directory service, it would also be possible to consider data, which are available over one or two hops (Edge Computing Nodes) forward or backward. Therefore, the logs between some Edge Computing Nodes would need to be synchronized.

12.3 Real World Experiments

Up to this point we only performed one real world experiment using a single car to validate our concept for edge computing. However, this was a very small and conceptual experiment and it is required to do more experiments here to test the stability and the scalability of our system. All other experiments were run in simulation. Even if we tried to simulate our example experiments in such a way that they are close to reality, usually when moving from simulation to reality some complications occur. For example, in our simulations we did not consider node or link failures or any other problems. Due to the lack of infrastructure available to perform these real world experiments, we needed to rely on the simulations. But since the simulation results are promising, it would be interesting to see the NFN system in real world.

12.4 Planning in Swarm Scenarios

In general, the CCN communication model is designed for infrastructure-rich scenarios. There are approaches and concepts how to apply CCN to *peer-2-peer* networks [LHB⁺13].

However, in this thesis, the we covered resolution strategies for rather static scenarios, where exactly one node executes a computation and where it was ok that some nodes changed their location, but not too many.

For example, the Plan-based resolution strategy is working around minor network changes during the execution by relying on the routing system and finding other nodes to execute the computation by sacrificing a bit of the optimal execution costs.

However, a Plan-based resolution strategy of NFN could also be used to run computation in loose environments such as swarms by changing the planning system to the needs of a swarm environment. Therefore, a plan would define which node of a swarm executes which part of a computation and also which node replicates which particular computation, since replications are very important in loose environments where nodes join and leave. A plan is distributed over the entire swarm. For the plan itself the same replication is required as for the computation itself, since losing a node with a plan would require re-planning.

In such an environment the planning process would be different. It would consist of collection information about which nodes are available and willing to help and then of optimally distributing and replicating the individual task. In general, CCN is not designed for swarm networks, thus for swarm computations another approach would be to replace the network layer with a network system which better fits the needs for swarm environments, while keeping the NFN layer to describe workflows and to perform computations.

PART V

Appendix

A

Raw Results of the Data Center Simulations

In the following we show some figures visualizing the full results of the data center simulation individually (see Evaluation, Section 9.2). Thereby, we use round bar graph, where each bar graph represents an individual run of the computation (overall 1000 runs, so 1000 bars). We plot all runs as an individual bar, so they fill up the round bar graph. The length of the bars is normalized, so one can see and compare the visualized share of the forwarding, the data transfer and the execution time.

Table A.1 gives an overview over the figures in this chapter:

Table A.1 Overview over Figures showing the Raw Results of the Simulations of the Data Center.

Resolution Strategy	Figure with small data center results	Figure with big data center results
To-Data-First	A.1	A.2
Map-Reduce	A.3	A.4
Plan based	A.5	A.6
Template based	A.7	A.8

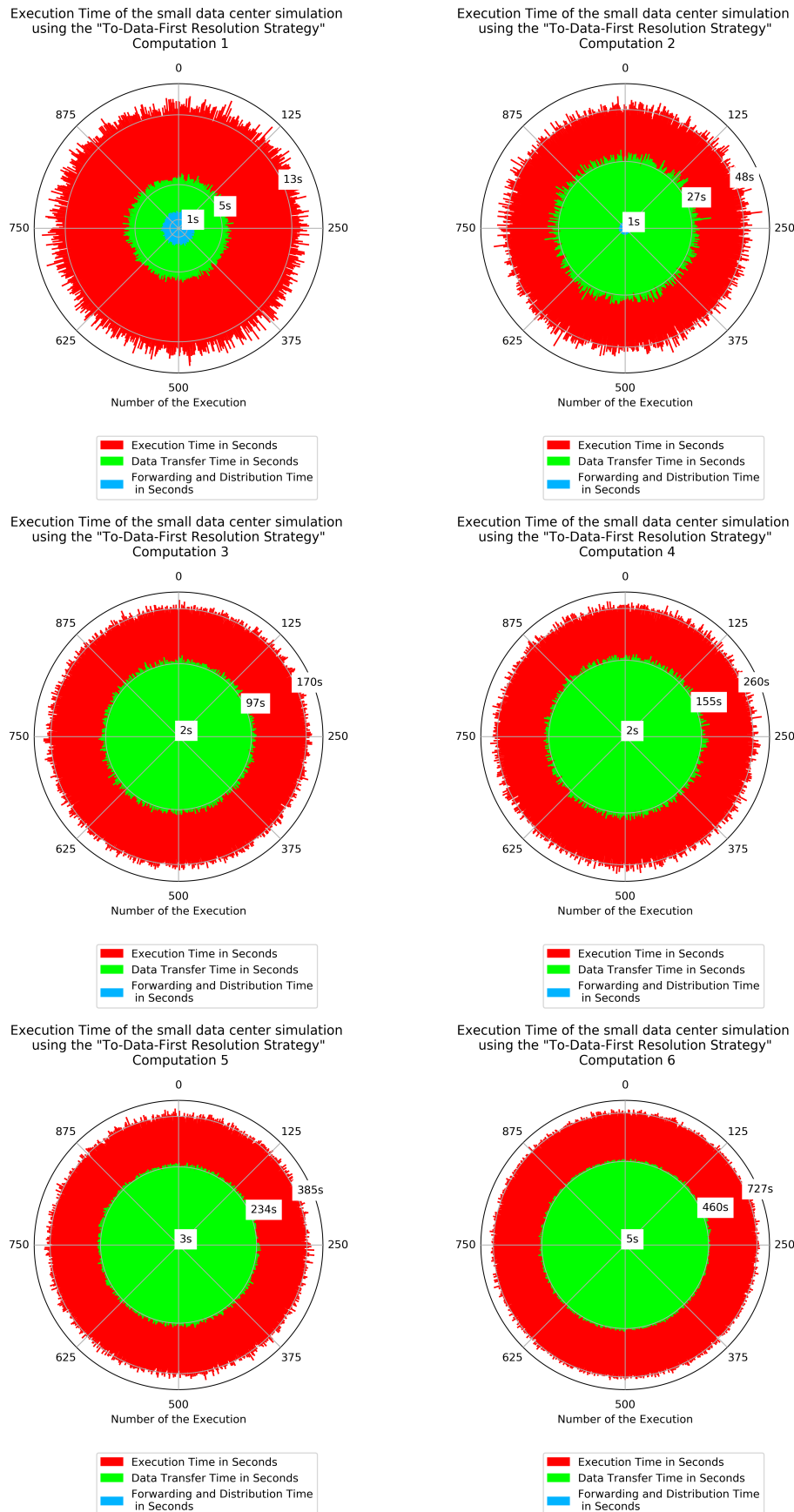
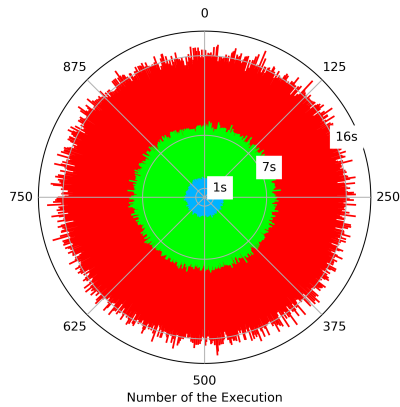


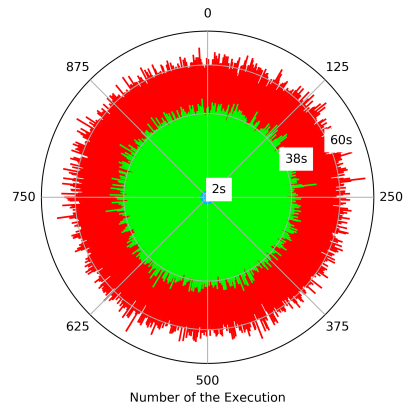
Figure A.1 Raw Results of the small data center Simulation with the *To-Data-First* resolution strategy.

These figures visualize the raw resulting data of the simulation. Each bar represents a single run. The overall time is the sum of the three individual parts. The time values are the mean times of each part of the overall execution.

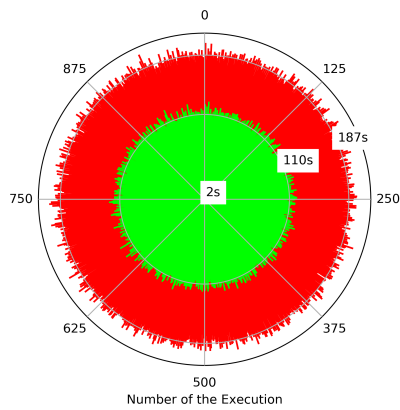
Execution Time of the big data center simulation
using the "To-Data-First Resolution Strategy"
Computation 1



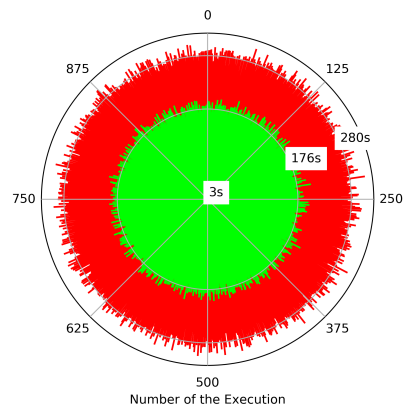
Execution Time of the big data center simulation
using the "To-Data-First Resolution Strategy"
Computation 2



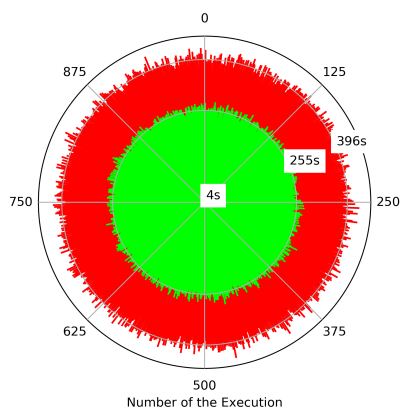
Execution Time of the big data center simulation
using the "To-Data-First Resolution Strategy"
Computation 3



Execution Time of the big data center simulation
using the "To-Data-First Resolution Strategy"
Computation 4



Execution Time of the big data center simulation
using the "To-Data-First Resolution Strategy"
Computation 5



Execution Time of the big data center simulation
using the "To-Data-First Resolution Strategy"
Computation 6

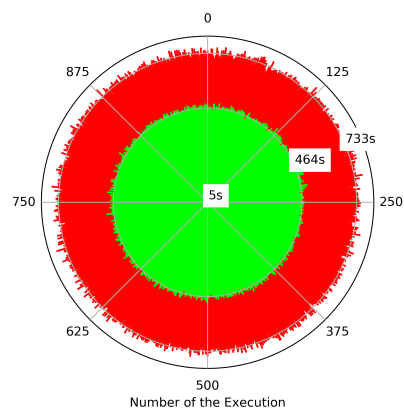


Figure A.2 Raw Results of the big data center Simulation with the *To-Data-First resolution strategy*.

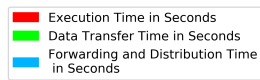
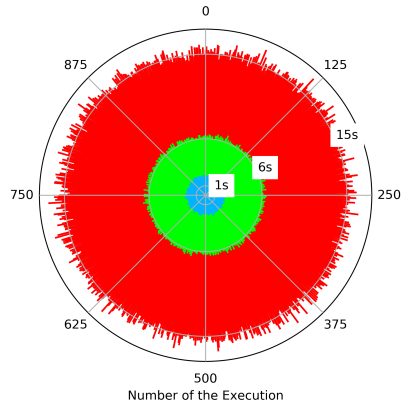
These figures visualize the raw resulting data of the simulation. Each bar represents a single run. The overall time is the sum of the three individual parts. The time values are the mean times of each part of the overall execution.



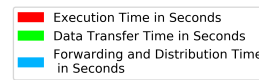
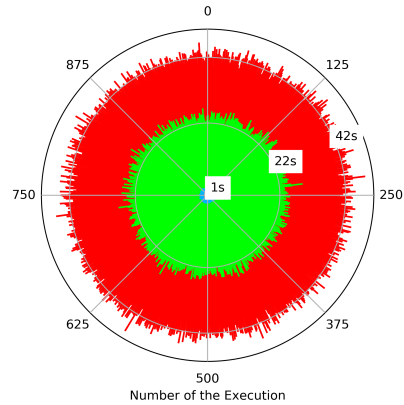
Figure A.3 Raw Results of the small data center Simulation with the *Map-Reduce resolution strategy*.

These figures visualize the raw resulting data of the simulation. Each bar represents a single run. The overall time is the sum of the three individual parts. The time values are the mean times of each part of the overall execution.

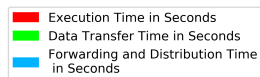
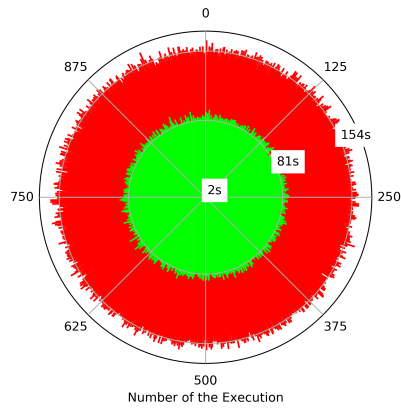
Execution Time of the big data center simulation
using the "Map-Reduce Resolution Strategy"
Computation 1



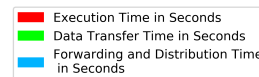
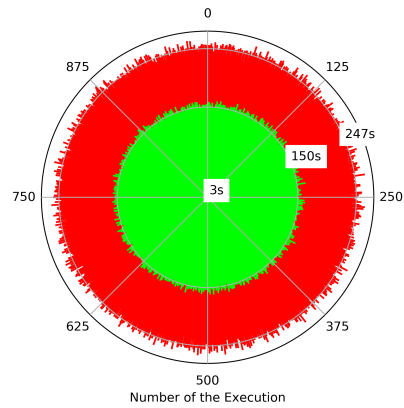
Execution Time of the big data center simulation
using the "Map-Reduce Resolution Strategy"
Computation 2



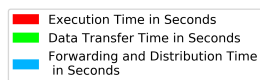
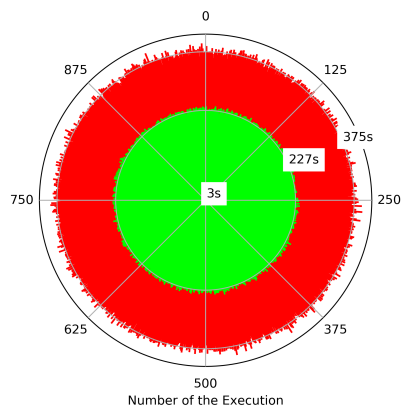
Execution Time of the big data center simulation
using the "Map-Reduce Resolution Strategy"
Computation 3



Execution Time of the big data center simulation
using the "Map-Reduce Resolution Strategy"
Computation 4



Execution Time of the big data center simulation
using the "Map-Reduce Resolution Strategy"
Computation 5



Execution Time of the big data center simulation
using the "Map-Reduce Resolution Strategy"
Computation 6

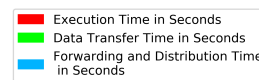
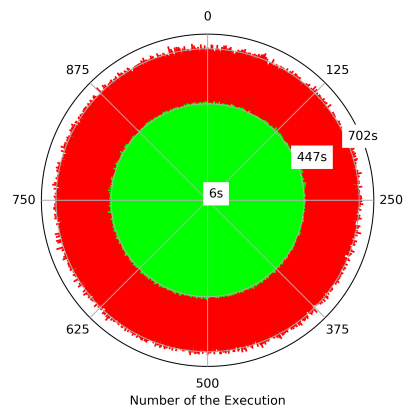


Figure A.4 Raw Results of the big data center Simulation with the *Map-Reduce resolution strategy*.

These figures visualize the raw resulting data of the simulation. Each bar represents a single run. The overall time is the sum of the three individual parts. The time values are the mean times of each part of the overall execution.

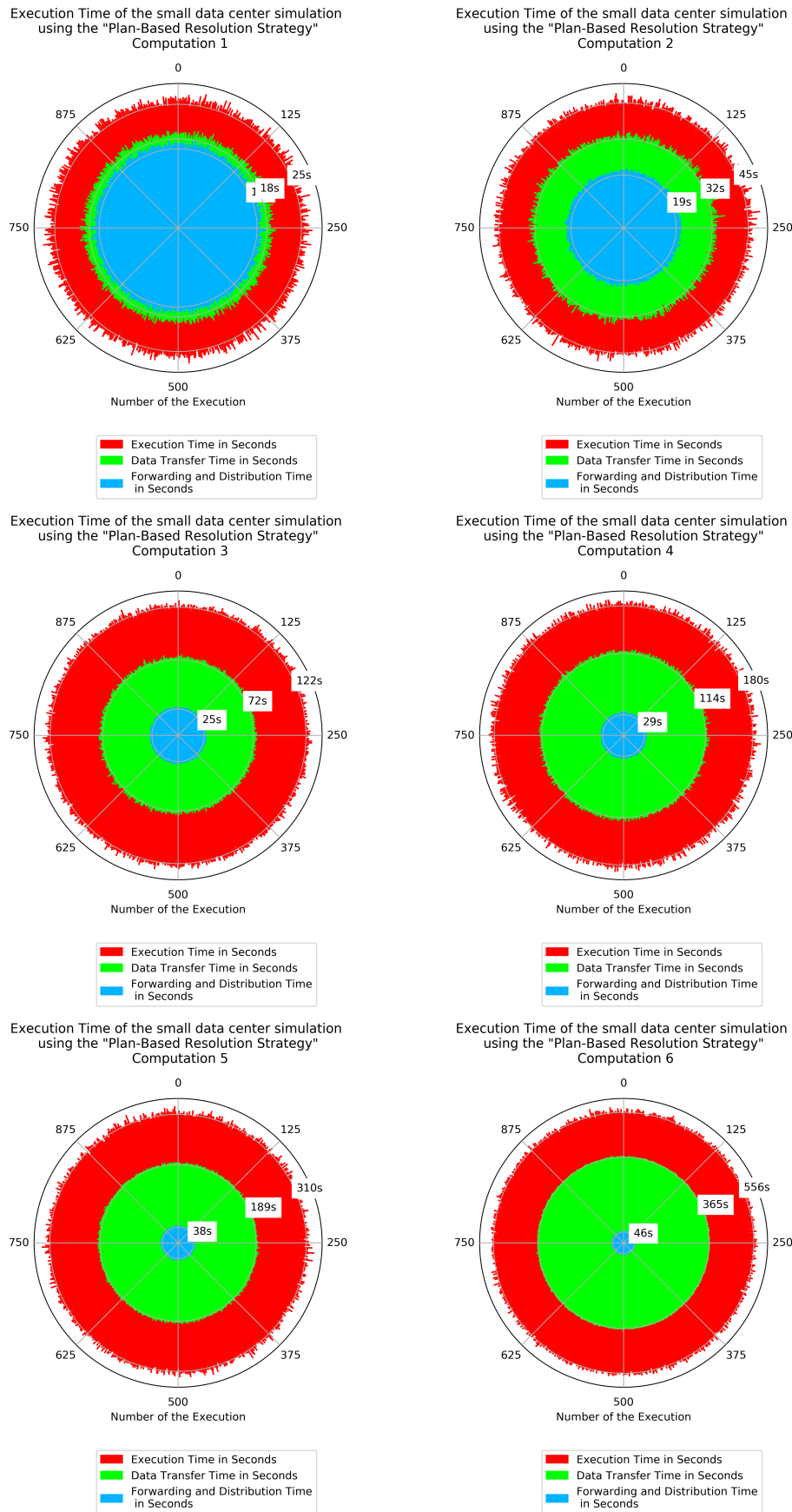


Figure A.5 Raw Results of the small data center Simulation with the *Plan-Based resolution strategy*.

These figures visualize the raw resulting data of the simulation. Each bar represents a single run. The overall time is the sum of the three individual parts. The time values are the mean times of each part of the overall execution.

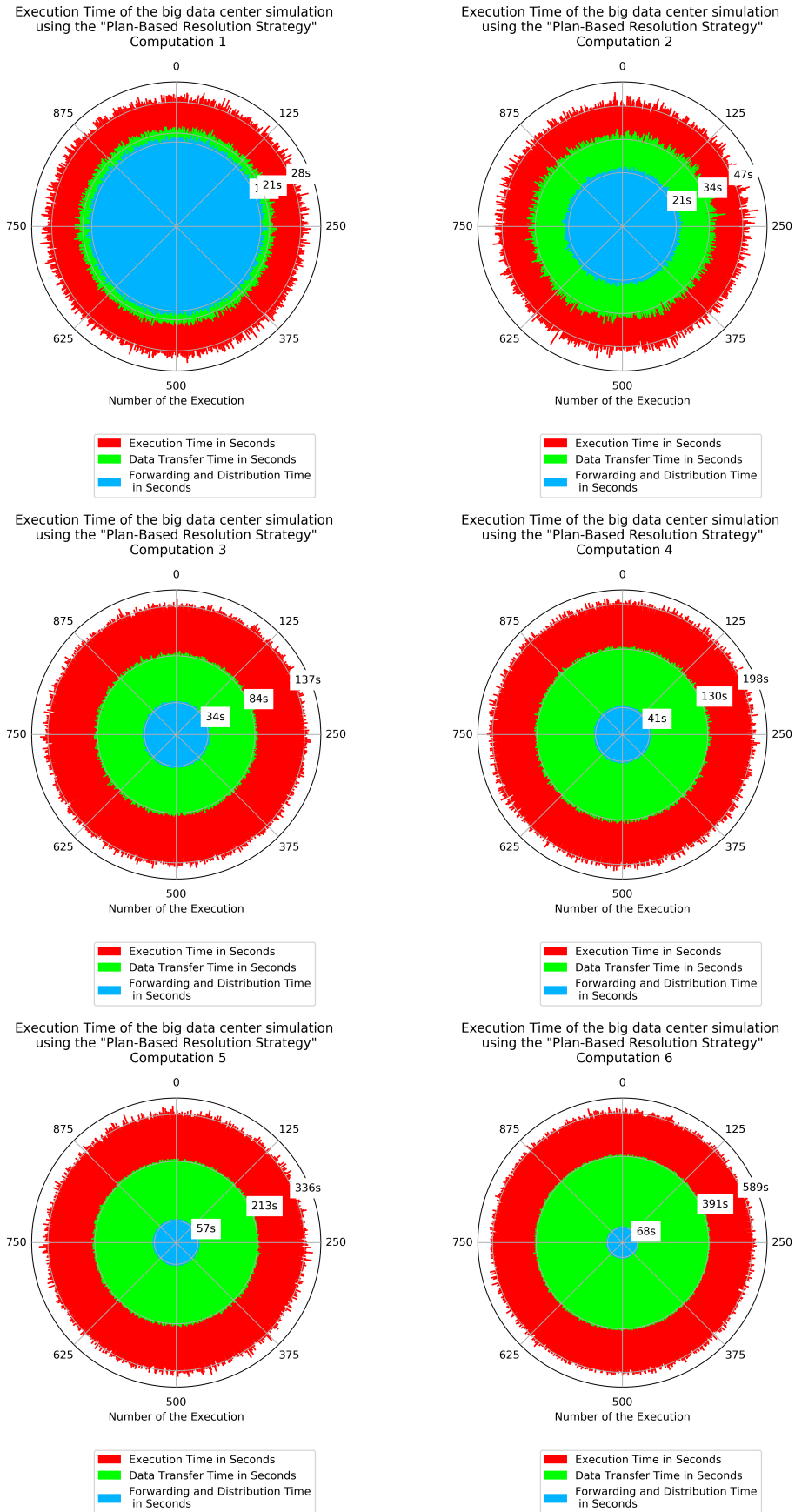


Figure A.6 Raw Results of the big data center Simulation with the *Plan-Based resolution strategy*.

These figures visualize the raw resulting data of the simulation. Each bar represents a single run. The overall time is the sum of the three individual parts. The time values are the mean times of each part of the overall execution.

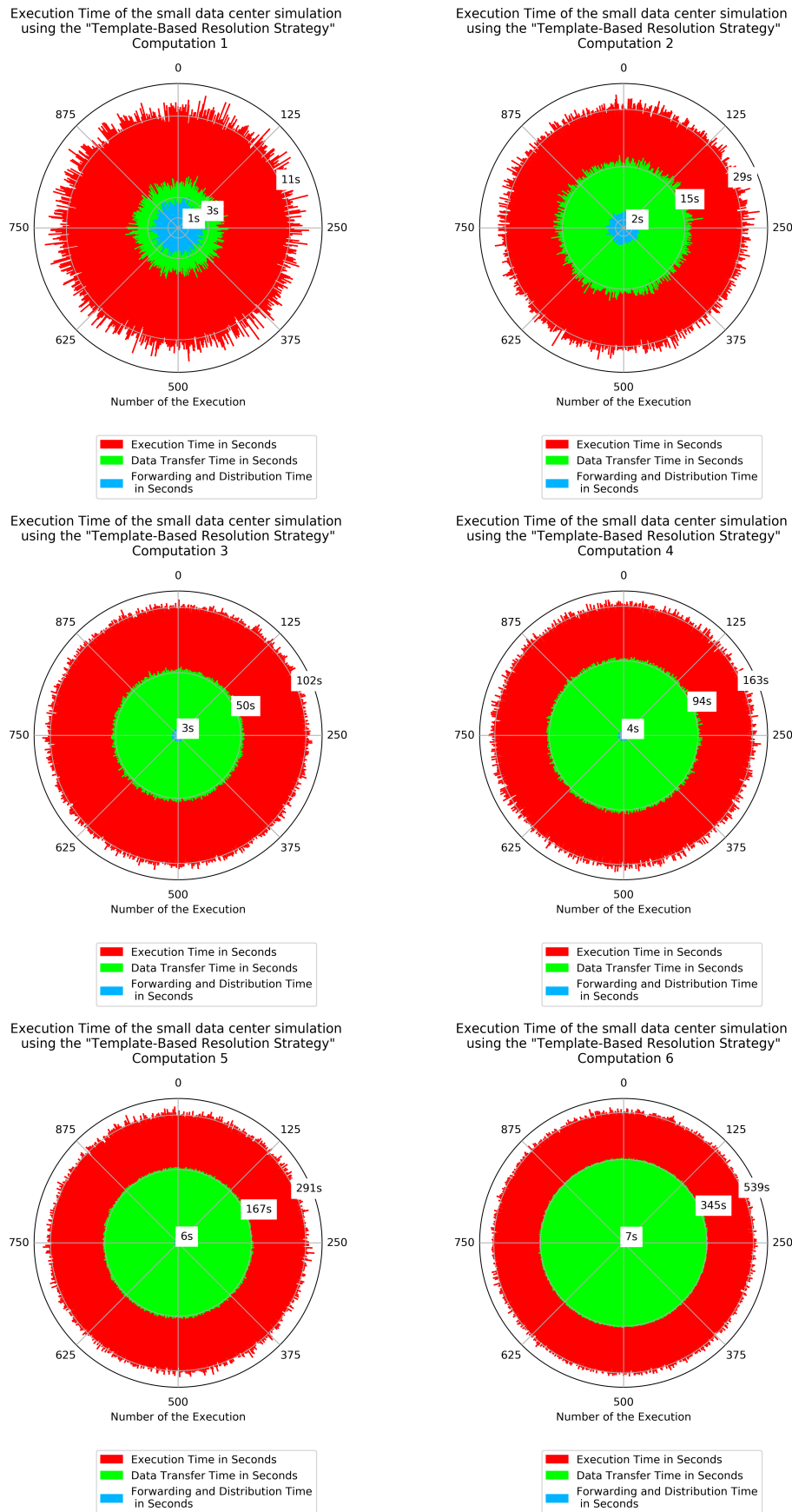




Figure A.8 Raw Results of the big data center Simulation with the *Template-Based resolution strategy*.

These figures visualize the raw resulting data of the simulation. Each bar represents a single run. The overall time is the sum of the three individual parts. The time values are the mean times of each part of the overall execution.

B

Percentage of Runtime Parts of the Data Center Simulations

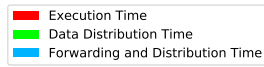
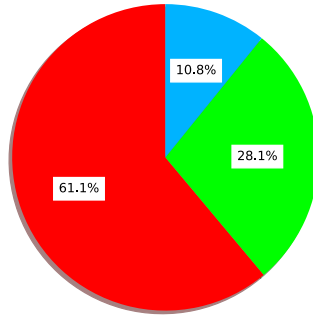
In the following we visualize the percentages of the *forwarding and distribution*-, the *data transfer*-, and the *execution* time as percentage of the overall time to resolve a computation, to show for which *resolution strategy* which part requires the largest amount of time and how this changes for longer computations (computation 1 has the smallest, computation 6 the longest execution time).

Table B.1 summarizes the figures in this chapter.

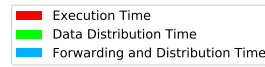
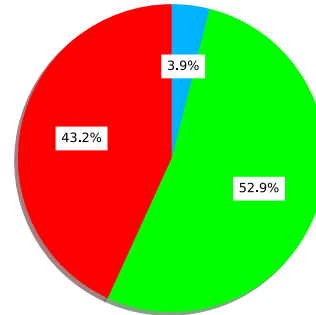
Table B.1 Overview over Figures showing the Percentage on the Overall Execution Time of the Simulations of the Data Center.

Resolution Strategy	Figure with small data center results	Figure with big data center results
To-Data-First	B.1	B.2
Map-Reduce	B.3	B.4
Plan based	B.5	B.6
Template based	B.7	B.8

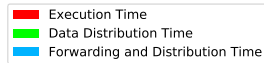
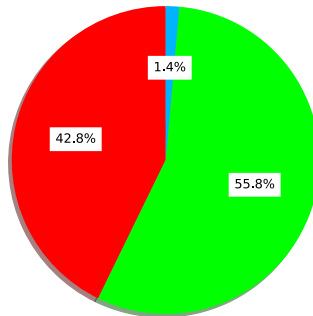
Mean Percentage Times of the small data center simulation using the "To-Data-First Resolution Strategy" Computation 1



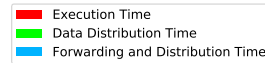
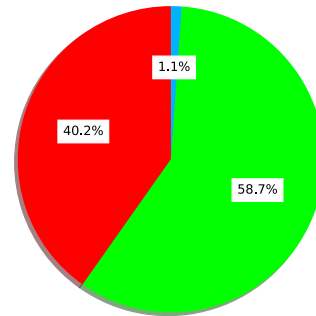
Mean Percentage Times of the small data center simulation using the "To-Data-First Resolution Strategy" Computation 2



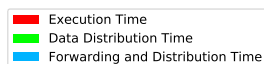
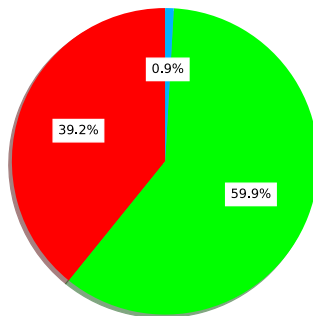
Mean Percentage Times of the small data center simulation using the "To-Data-First Resolution Strategy" Computation 3



Mean Percentage Times of the small data center simulation using the "To-Data-First Resolution Strategy" Computation 4



Mean Percentage Times of the small data center simulation using the "To-Data-First Resolution Strategy" Computation 5



Mean Percentage Times of the small data center simulation using the "To-Data-First Resolution Strategy" Computation 6

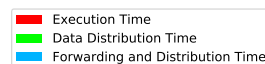
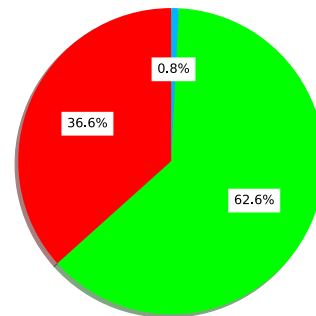
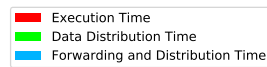
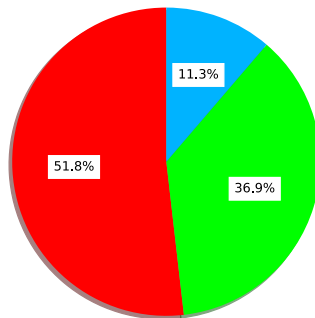


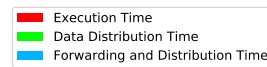
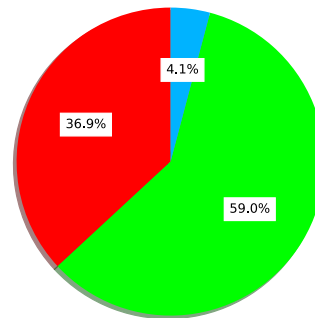
Figure B.1 Mean Percentage of the Runtime of the small data center Simulation with the *To-Data-First* resolution strategy.

Mean Percentage of the *Forwarding and Distribution Time*, the *Time for Data Transfers* and the *Time for Execution and Delivery of the Result* for the small data center Simulation with the *To-Data-First* resolution strategy.

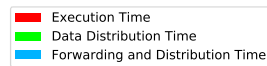
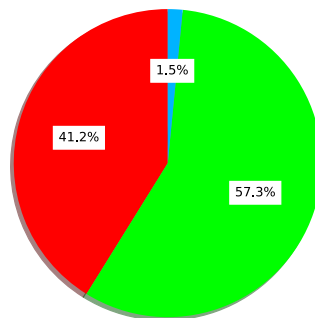
Mean Percentage Times of the big data center simulation
using the "To-Data-First Resolution Strategy"
Computation 1



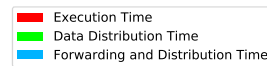
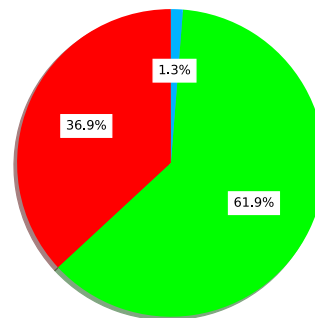
Mean Percentage Times of the big data center simulation
using the "To-Data-First Resolution Strategy"
Computation 2



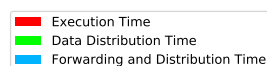
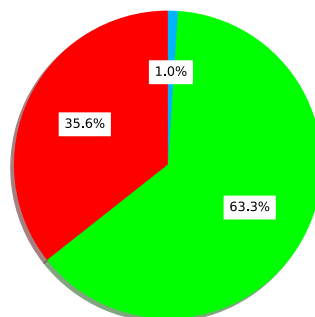
Mean Percentage Times of the big data center simulation
using the "To-Data-First Resolution Strategy"
Computation 3



Mean Percentage Times of the big data center simulation
using the "To-Data-First Resolution Strategy"
Computation 4



Mean Percentage Times of the big data center simulation
using the "To-Data-First Resolution Strategy"
Computation 5



Mean Percentage Times of the big data center simulation
using the "To-Data-First Resolution Strategy"
Computation 6

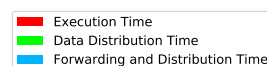
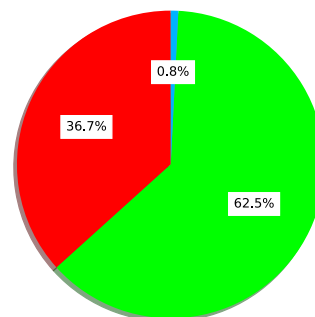
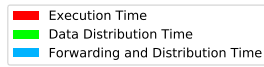
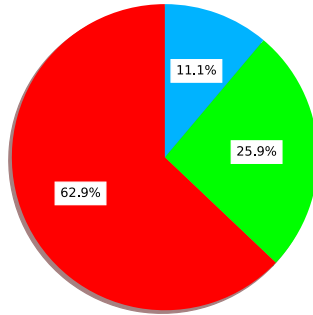


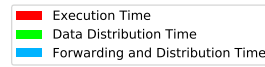
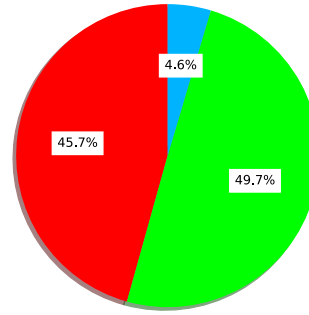
Figure B.2 Mean Percentage of the Runtime of the big data center Simulation with the *To-Data-First* resolution strategy.

Mean Percentage of the *Forwarding and Distribution Time*, the *Time for Data Transfers* and the *Time for Execution and Delivery of the Result* for the big data center Simulation with the *To-Data-First* resolution strategy.

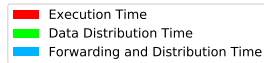
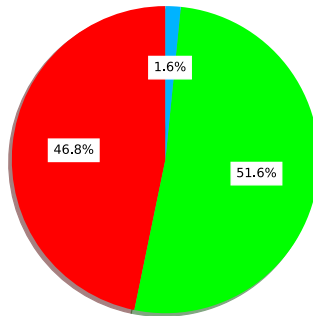
Mean Percentage Times of the small data center simulation using the "Map-Reduce Resolution Strategy" Computation 1



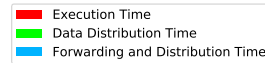
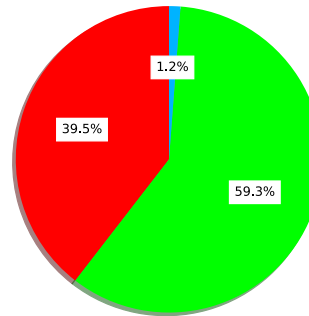
Mean Percentage Times of the small data center simulation using the "Map-Reduce Resolution Strategy" Computation 2



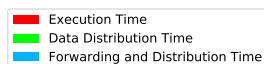
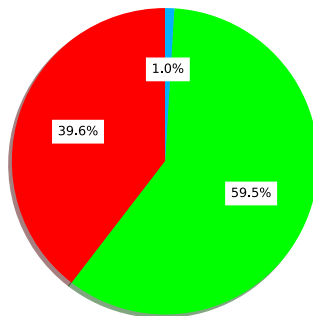
Mean Percentage Times of the small data center simulation using the "Map-Reduce Resolution Strategy" Computation 3



Mean Percentage Times of the small data center simulation using the "Map-Reduce Resolution Strategy" Computation 4



Mean Percentage Times of the small data center simulation using the "Map-Reduce Resolution Strategy" Computation 5



Mean Percentage Times of the small data center simulation using the "Map-Reduce Resolution Strategy" Computation 6

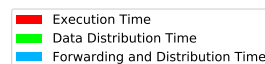
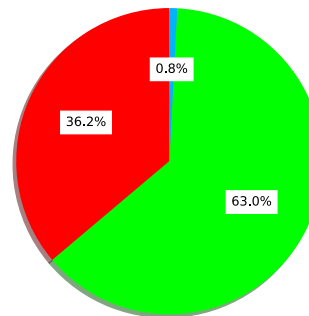
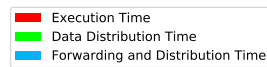
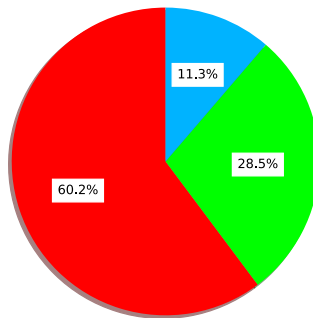


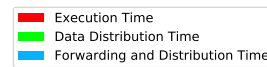
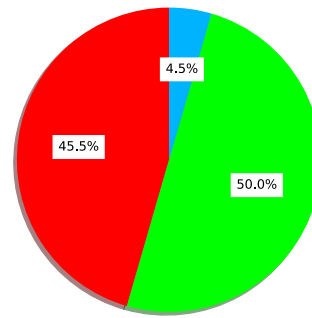
Figure B.3 Mean Percentage of the Runtime of the small data center Simulation with the *Map-Reduce* resolution strategy.

Mean Percentage of the *Forwarding and Distribution Time*, the *Time for Data Transfers* and the *Time for Execution and Delivery of the Result* for the small data center Simulation with the *Map-Reduce* resolution strategy.

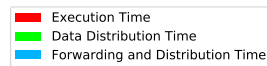
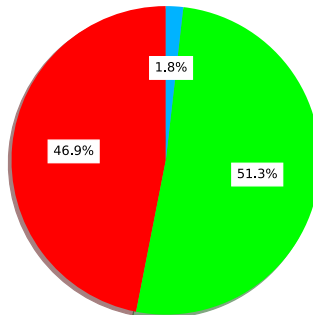
Mean Percentage Times of the big data center simulation
using the "Map-Reduce Resolution Strategy"
Computation 1



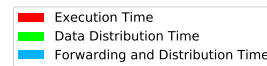
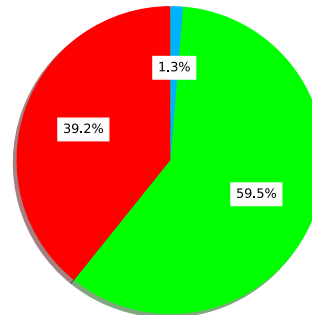
Mean Percentage Times of the big data center simulation
using the "Map-Reduce Resolution Strategy"
Computation 2



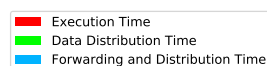
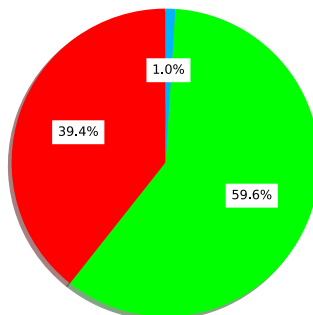
Mean Percentage Times of the big data center simulation
using the "Map-Reduce Resolution Strategy"
Computation 3



Mean Percentage Times of the big data center simulation
using the "Map-Reduce Resolution Strategy"
Computation 4



Mean Percentage Times of the big data center simulation
using the "Map-Reduce Resolution Strategy"
Computation 5



Mean Percentage Times of the big data center simulation
using the "Map-Reduce Resolution Strategy"
Computation 6

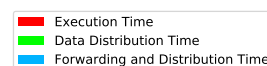
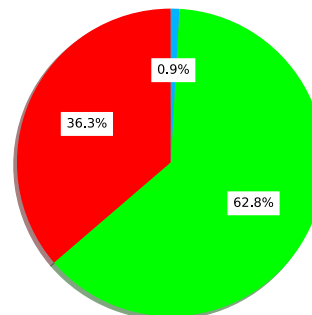
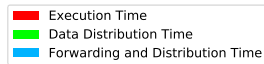
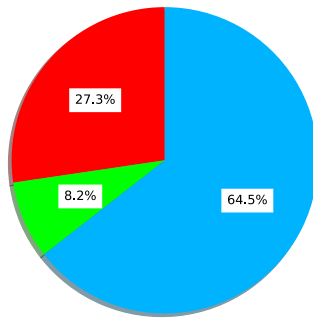


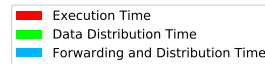
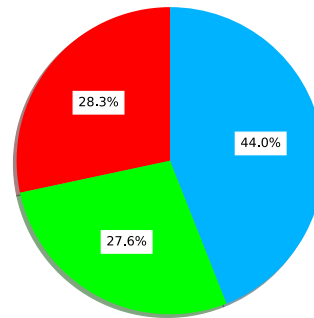
Figure B.4 Mean Percentage of the Runtime of the big data center Simulation with the *Map-Reduce* resolution strategy.

Mean Percentage of the *Forwarding and Distribution Time*, the *Time for Data Transfers* and the *Time for Execution and Delivery of the Result* for the big data center Simulation with the *Map-Reduce* resolution strategy.

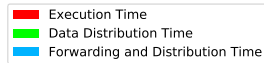
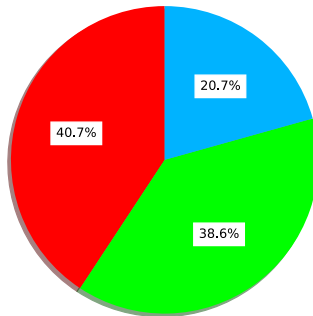
Mean Percentage Times of the small data center simulation using the "Plan-Based Resolution Strategy" Computation 1



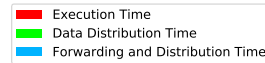
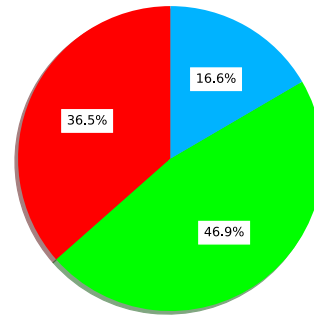
Mean Percentage Times of the small data center simulation using the "Plan-Based Resolution Strategy" Computation 2



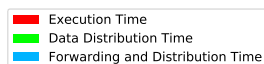
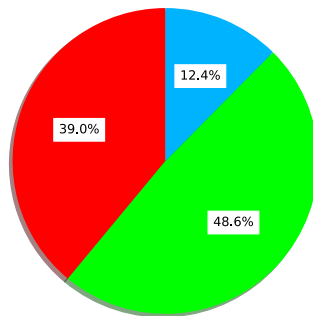
Mean Percentage Times of the small data center simulation using the "Plan-Based Resolution Strategy" Computation 3



Mean Percentage Times of the small data center simulation using the "Plan-Based Resolution Strategy" Computation 4



Mean Percentage Times of the small data center simulation using the "Plan-Based Resolution Strategy" Computation 5



Mean Percentage Times of the small data center simulation using the "Plan-Based Resolution Strategy" Computation 6

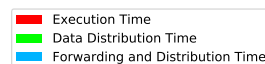
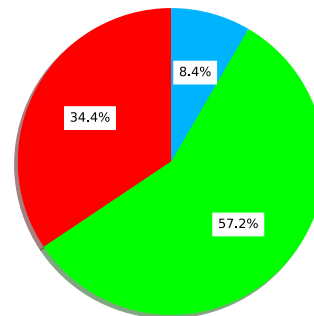
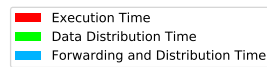
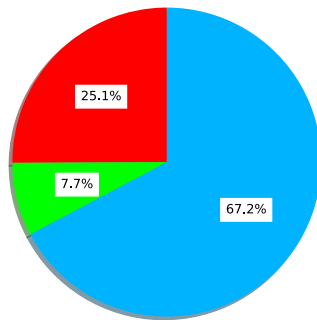


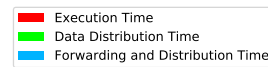
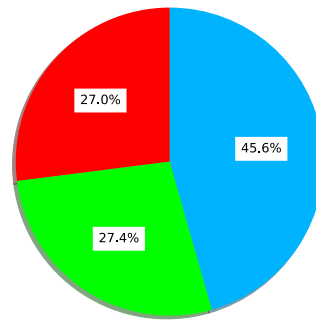
Figure B.5 Mean Percentage of the Runtime of the small data center Simulation with the *Plan-Based resolution strategy*.

Mean Percentage of the *Forwarding and Distribution Time*, the *Time for Data Transfers* and the *Time for Execution and Delivery of the Result* for the small data center Simulation with the *Plan-Based resolution strategy*.

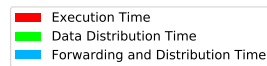
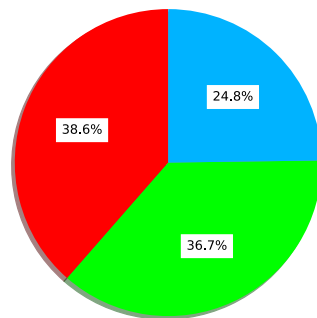
Mean Percentage Times of the big data center simulation using the "Plan-Based Resolution Strategy" Computation 1



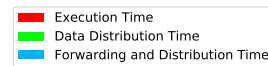
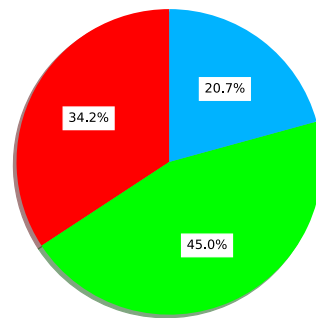
Mean Percentage Times of the big data center simulation using the "Plan-Based Resolution Strategy" Computation 2



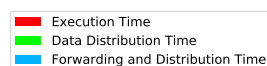
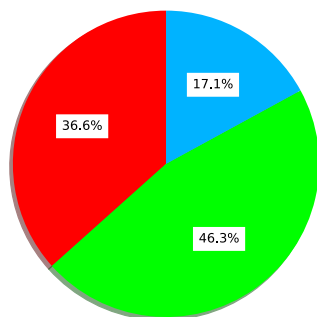
Mean Percentage Times of the big data center simulation using the "Plan-Based Resolution Strategy" Computation 3



Mean Percentage Times of the big data center simulation using the "Plan-Based Resolution Strategy" Computation 4



Mean Percentage Times of the big data center simulation using the "Plan-Based Resolution Strategy" Computation 5



Mean Percentage Times of the big data center simulation using the "Plan-Based Resolution Strategy" Computation 6

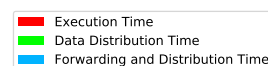
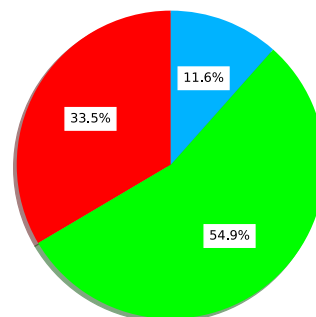
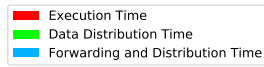
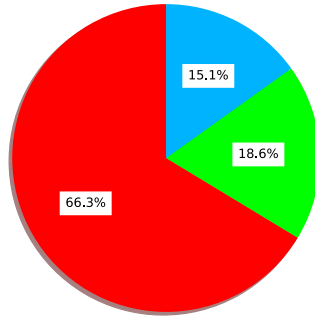


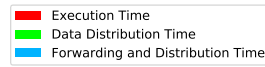
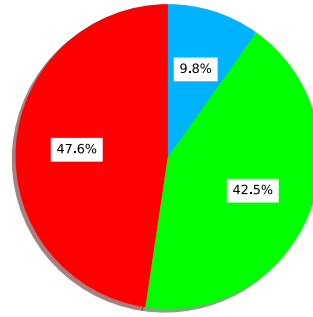
Figure B.6 Mean Percentage of the Runtime of the big data center Simulation with the *Plan-Based resolution strategy*.

Mean Percentage of the *Forwarding and Distribution Time*, the *Time for Data Transfers* and the *Time for Execution and Delivery of the Result* for the big data center Simulation with the *Plan-Based resolution strategy*.

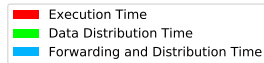
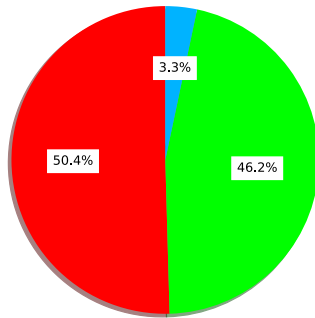
Mean Percentage Times of the small data center simulation using the "Template-Based Resolution Strategy" Computation 1



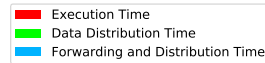
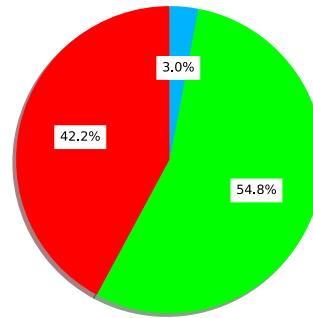
Mean Percentage Times of the small data center simulation using the "Template-Based Resolution Strategy" Computation 2



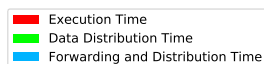
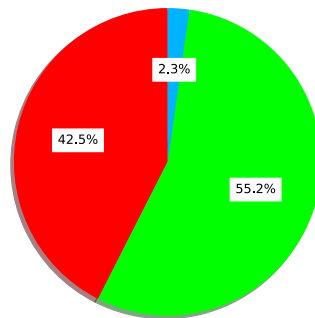
Mean Percentage Times of the small data center simulation using the "Template-Based Resolution Strategy" Computation 3



Mean Percentage Times of the small data center simulation using the "Template-Based Resolution Strategy" Computation 4



Mean Percentage Times of the small data center simulation using the "Template-Based Resolution Strategy" Computation 5



Mean Percentage Times of the small data center simulation using the "Template-Based Resolution Strategy" Computation 6

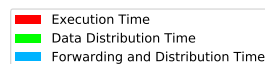
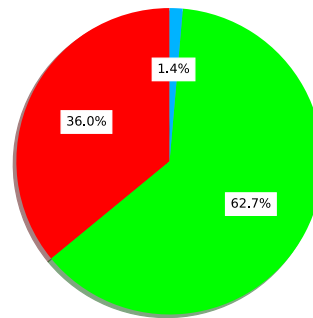
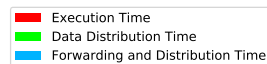
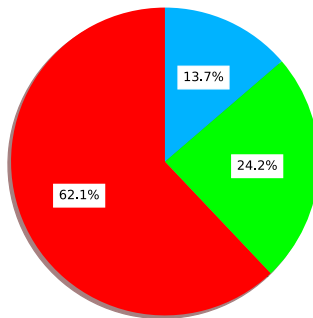


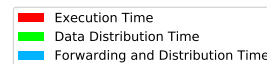
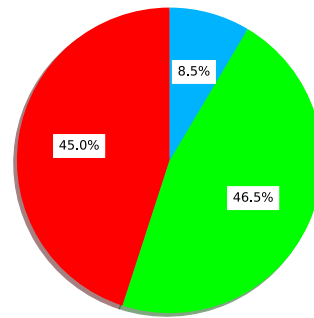
Figure B.7 Mean Percentage of the Runtime of the small data center Simulation with the *Template-based resolution strategy*.

Mean Percentage of the *Forwarding and Distribution Time*, the *Time for Data Transfers* and the *Time for Execution and Delivery of the Result* for the small data center Simulation with the *Template-Based resolution strategy*.

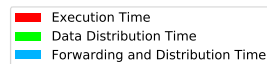
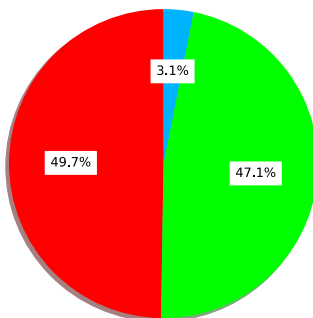
Mean Percentage Times of the big data center simulation
using the "Template-Based Resolution Strategy"
Computation 1



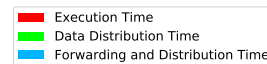
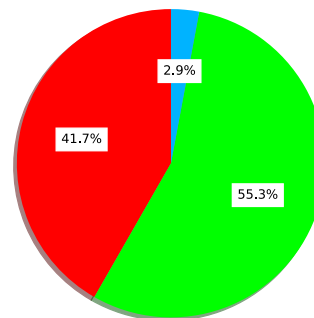
Mean Percentage Times of the big data center simulation
using the "Template-Based Resolution Strategy"
Computation 2



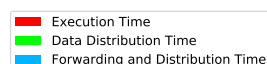
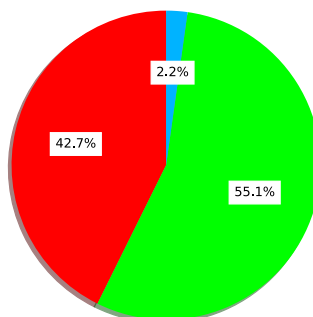
Mean Percentage Times of the big data center simulation
using the "Template-Based Resolution Strategy"
Computation 3



Mean Percentage Times of the big data center simulation
using the "Template-Based Resolution Strategy"
Computation 4



Mean Percentage Times of the big data center simulation
using the "Template-Based Resolution Strategy"
Computation 5



Mean Percentage Times of the big data center simulation
using the "Template-Based Resolution Strategy"
Computation 6

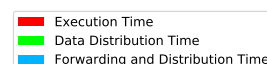
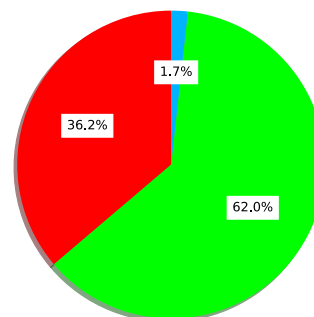


Figure B.8 Mean Percentage of the Runtime of the big data center Simulation with the *Template-Based resolution strategy*.

Mean Percentage of the *Forwarding and Distribution Time*, the *Time for Data Transfers* and the *Time for Execution and Delivery of the Result* for the big data center Simulation with the *Template-Based resolution strategy*.

Bibliography

- [AGH⁺15] V. K. Adhikari, Y. Guo, F. Hao, V. Hilt, Z. Zhang, M. Varvello, and M. Steiner. Measurement study of netflix, hulu, and a tale of three cdns. *IEEE/ACM Transactions on Networking*, 23(6):1984–1997, 2015. ISSN: 1558-2566.
- [ADI⁺12] Bengt Ahlgren, Christian Dannewitz, Claudio Imbrenda, Dirk Kutscher, and Borje Ohlman. A survey of information-centric networking. *IEEE Communications Magazine*, 50(7), 2012.
- [Ama18] Amazon. Amazon lambda. <https://aws.amazon.com/lambda/features/>, 2018. [Online; accessed 24-Nov-2018].
- [ACC⁺13] Giuseppe Araniti, Claudia Campolo, Massimo Condoluci, Antonio Iera, and Antonella Molinaro. Lte for vehicular networking: a survey. *IEEE communications magazine*, 51(5):148–157, 2013.
- [Ari18] Chad Arimura. Functions vs containers. <https://medium.com/oracledevs/containers-vs-functions-51c879216b97>, 2018. [Online; accessed 24-Nov-2018].
- [BCC⁺17] Ioana Baldini, Paul Castro, Kerry Chang, Perry Cheng, Stephen Fink, Vatche Ishakian, Nick Mitchell, Vinod Muthusamy, Rodric Rabbah, Aleksander Slominski, et al. Serverless computing: current trends and open problems. In *Research Advances in Cloud Computing*, pages 1–20. Springer, 2017.
- [BMZ⁺12] Flavio Bonomi, Rodolfo Milito, Jiang Zhu, and Sateesh Addepalli. Fog computing and its role in the internet of things. In *Proceedings of the first edition of the MCC workshop on Mobile cloud computing*, pages 13–16. ACM, 2012.
- [BHH⁺11] Torsten Braun, Volker Hilt, Markus Hofmann, Ivica Rimac, Moritz Steiner, and Matteo Varvello. Service-centric networking. In *2011 IEEE International Conference on Communications Workshops (ICC)*, pages 1–6. IEEE, 2011.
- [Bur17] Jeff Burke. Browsing an augmented reality with named data networking. In *International Conference on Computer Communication and Networks (ICCCN), International Conference on Computer Communication and Networks (ICCCN)*, 2017.

- [Cha98] Surajit Chaudhuri. An overview of query optimization in relational systems. In *Proceedings of the seventeenth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems*, pages 34–43. ACM, 1998.
- [CPR18] Continental-Press-Release. Leading automotive, telecom and its companies successfully carry out first cellular v2x trial in japan. <https://www.continental-corporation.com/en/press/press-releases/2018-12-13-japan-155774>, 2018. [Online; accessed 22-Dec-2018], released: 2018-12-13.
- [DKO⁺13] Christian Dannewitz, Dirk Kutscher, Börje Ohlman, Stephen Farrell, Bengt Ahlgren, and Holger Karl. Network of information (net-inf) – an information-centric networking architecture. *Computer Communications*, 36(7):721–735, 2013. ISSN: 0140-3664.
- [Dek⁺59] JCE Dekker et al. Martin davis, computability and unsolvability. *Bulletin of the American Mathematical Society*, 65(4):70–71, 1959.
- [Dzo18] Dzone. 4 use cases of serverless architecture. <https://dzone.com/articles/4-use-cases-of-serverless-architecture>, 2018. [Online; accessed 26-Nov-2018].
- [ETS08] ETSI. Its-g5 etsi en 302 571: : intelligent transport systems (its); radiocommunications equipment operating in the 5 855 mhz to 5 925 mhz frequency band. In *Harmonized European Standard*. ETSI, 2008.
- [FLRK80] Michael Florian, Jan Karel Lenstra, and AHG Rinnooy Kan. Deterministic production planning: algorithms and complexity. *Management science*, 26(7):669–679, 1980.
- [GTU⁺13] Paolo Gasti, Gene Tsudik, Ersin Uzun, and Lixia Zhang. Dos and ddos in named data networking. In *Computer Communications and Networks (ICCCN), 2013 22nd International Conference on*, pages 1–7. IEEE, 2013.
- [Ger12] Mario Gerla. Vehicular cloud computing. In *Ad Hoc Networking Workshop (Med-Hoc-Net), 2012 The 11th Annual Mediterranean*, pages 152–155. IEEE, 2012.

- [GSK⁺11] Ali Ghodsi, Scott Shenker, Teemu Koponen, Ankit Singla, Barath Raghavan, and James Wilcox. Information-centric networking: seeing the forest for the trees. In *Proceedings of the 10th ACM Workshop on Hot Topics in Networks, HotNets-X*, 1:1–1:6, Cambridge, Massachusetts. ACM, 2011. ISBN: 978-1-4503-1059-8.
- [GMAF⁺19] Clement Godard, Oisin Mac Aodha, Michael Firman, and Gabriel J Brostow. Monodepth. <https://github.com/nianticlabs/monodepth2>, 2019. [Online; accessed 20-Nov-2019].
- [GY19] Andrea Goldsmith and Edmund Yeh. Optimizing computation scheduling, caching, and forwarding at the edge. In 2019.
- [Goo18] Google. Google cloud functions. <https://cloud.google.com/functions/>, 2018. [Online; accessed 24-Nov-2018].
- [GMS⁺18] Dennis Grewe, Claudio Marxer, Christopher Scherb, Marco Wagner, and Christian Tschudin. A network stack for computation-centric vehicular networking. In *Proceedings of the 5th ACM Conference on Information-Centric Networking*, pages 208–209. ACM, 2018.
- [GWA⁺17] Dennis Grewe, Marco Wagner, Mayutan Arumathurai, Ioannis Psaras, and Dirk Kutscher. Information-centric mobile edge computing for connected vehicle environments: challenges and research directions. In *Proceedings of the Workshop on Mobile Edge Communications*, pages 7–12. ACM, 2017.
- [GSS⁺18] C Gundogan, C Scherb, T Schmidt, M Waehlich, P Kietzmann, C Marxer, and C Tschudin. Icn adaptation to lowpan networks (icn lowpan) draft-gundogan-icnrg-ccnlowpan-02, 2018.
- [HBF⁺11] Kerry Hinton, Jayant Baliga, Michael Feng, Robert Ayre, and Rodney S Tucker. Power consumption and energy efficiency in the internet. *IEEE Network*, 25(2):6–12, 2011.
- [IBM18] IBM. Ibm cloud functions. <https://console.bluemix.net/openwhisk/>, 2018. [Online; accessed 24-Nov-2018].
- [IKAM17] Hatem Ibn-Khedher, Hossam Afifi, and Hassine Moun gla. Optimal hadoop over icn placement algorithm for networking and distributed computing. In *GLOBECOM 2017-2017 IEEE Global Communications Conference*, pages 1–6. IEEE, 2017.

- [JST⁺09] Van Jacobson, Diana K Smetters, James D Thornton, Michael F Plass, Nicholas H Briggs, and Rebecca L Braynard. Networking named content. In *Proceedings of the 5th international conference on Emerging networking experiments and technologies*, pages 1–12. ACM, 2009.
- [JD08] Daniel Jiang and Luca Delgrossi. Ieee 802.11 p: towards an international standard for wireless access in vehicular environments. In *Vehicular Technology Conference, 2008. VTC Spring 2008. IEEE*, pages 2036–2040. IEEE, 2008.
- [KCC⁺07] Teemu Koponen, Mohit Chawla, Byung-Gon Chun, Andrey Ermolinskiy, Kye Hyun Kim, Scott Shenker, and Ion Stoica. A data-oriented (and beyond) network architecture. In *Proceedings of the 2007 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications, SIGCOMM '07*, pages 181–192, Kyoto, Japan. ACM, 2007. ISBN: 978-1-59593-713-1.
- [KHO⁺18] Michał Król, Karim Habak, David Oran, Dirk Kutscher, and Ioannis Psaras. Rice: remote method invocation in icn. In *Proceedings of the 5th ACM Conference on Information-Centric Networking*. ACM, 2018.
- [KMO⁺19] Michał Król, Spyridon Mastorakis, David Oran, and Dirk Kutscher. Compute first networking: distributed computing meets icn. In *Proceedings of the 6th ACM Conference on Information-Centric Networking*, pages 67–77. ACM, 2019.
- [KP17] Michał Król and Ioannis Psaras. Nfaas: named function as a service. In *Proceedings of the 4th ACM Conference on Information-Centric Networking*, pages 134–144. ACM, 2017.
- [LVT10] Dmitrij Lagutin, Kari Visala, and Sasu Tarkoma. Publish/subscribe for internet: psirp perspective. *Future internet assembly*, 84, 2010.
- [Lea18] LeanBi. Fog und edge computing für industrien versus cloud. <https://leanbi.ch/blog/iot-und-predictive-analytics-fog-und-edge-computing-fur-industrien-versus-cloud-19-1-2018/>, 2018. [Online; accessed 28-Nov-2018].
- [LGG⁺17] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. Focal loss for dense object detection. In *Proceedings of the IEEE international conference on computer vision*, pages 2980–2988, 2017.

- [LHB⁺13] Jared Lindblom, M Huang, Jeff Burke, and Lixia Zhang. Filesync/ndn: peer-to-peer file sync over named data networking. *NDN, TR*, 12, 2013.
- [LSW⁺08] Christian Lochert, Björn Scheuermann, Christian Wewetzer, Andreas Luebke, and Martin Mauve. Data aggregation and roadside unit placement for a vanet traffic information system. In *Proceedings of the fifth ACM international workshop on VehiculAr Inter-NEtworking*, pages 58–65. ACM, 2008.
- [MST16] Claudio Marxer, Christopher Scherb, and Christian Tschudin. Access-controlled in-network processing of named data. In *Proceedings of the 3rd ACM Conference on Information-Centric Networking*, pages 77–82. ACM, 2016.
- [Med18] Medium. 3 simple use cases for leveraging serverless or faas “functions as a service” in your enterprise. <https://medium.com/devopslinks/3-simple-use-cases-for-leveraging-serverless-or-faas-functions-as-a-service-in-your-enterprise-409d431b7552>, 2018. [Online; accessed 27-Nov-2018].
- [Mic18] Microsoft. Microsoft azure functions. <https://azure.microsoft.com/en-us/services/functions/>, 2018. [Online; accessed 24-Nov-2018].
- [MHA⁺15] Shahid Mumtaz, Kazi Mohammed Saidul Huq, Muhammad Ikram Ashraf, Jonathan Rodriguez, Valdemar Monteiro, and Christos Politis. Cognitive vehicular communication for 5g. *IEEE Communications Magazine*, 53(7):109–117, 2015.
- [Nel16] Patrick Nelson. Just one autonomous car will use 4,000 gb of data/day. <https://www.networkworld.com/article/3147892/internet/one-autonomous-car-will-use-4000-gb-of-dataday.html>, 2016. [Online; accessed 27-Nov-2018].
- [Ora18] Oracle. Oracle fn. <https://developer.oracle.com/java/fn-project-introduction>, 2018. [Online; accessed 24-Nov-2018].
- [PIUB⁺17] G. Peralta, M. Iglesias-Urkia, M. Barcelo, R. Gomez, A. Moran, and J. Bilbao. Fog computing based efficient iot scheme for the industry 4.0. In *2017 IEEE International Workshop of Electronics, Control, Measurement, Signals and their Application to Mechatronics (ECMSM)*, volume 00, pages 1–6, 2017.

- [Pro18] Fn Project. Fn project. <https://github.com/fnproject>, 2018. [Online; accessed 24-Nov-2018].
- [RSM⁺13] Theodore S Rappaport, Shu Sun, Rimma Mayzus, Hang Zhao, Yaniv Azar, Kevin Wang, George N Wong, Jocelyn K Schulz, Mathew Samimi, and Felix Gutierrez Jr. Millimeter wave mobile communications for 5g cellular: it will work! *IEEE access*, 1(1):335–349, 2013.
- [RDG⁺16] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 779–788, 2016.
- [REK⁺06] Christian Ress, A ETERMAD, Detlef Kuck, and Marcus Boerger. Electronic horizon-supporting adas applications with predictive map data. In *PROCEEDINGS OF THE 13th ITS WORLD CONGRESS, LONDON, 8-12 OCTOBER 2006*, 2006.
- [Sat17] M. Satyanarayanan. The emergence of edge computing. *Computer*, 50(1):30–39, 2017. ISSN: 0018-9162.
- [SEM⁺19] Christopher Scherb, Samuel Emde, Claudio Marxer, and Christian Tschudin. Data upload in mobile edge computing over icn. In *Proc. of the IEEE Globecom 2019*. IEEE, 2019.
- [SFT17] Christopher Scherb, Balázs Faludi, and Christian Tschudin. Execution state management in named function networking. In *IFIP Networking Conference (IFIP Networking) and Workshops, 2017*, pages 1–6. IEEE, 2017.
- [SGW⁺18] Christopher Scherb, Dennis Grewe, Marco Wagner, and Christian Tschudin. Resolution strategies for networking the iot at the edge via named functions. In *2018 15th IEEE Annual Consumer Communications & Networking Conference (CCNC)*, pages 1–6. IEEE, 2018.
- [SJV12] Christopher Scherb, Christoph Jud, and Thomas Vetter. *Hybrid Image Registration of Electron Microscopy Sequences*, Universität Basel, 2012.
- [SMS⁺17] Christopher Scherb, Claudio Marxer, Urs Schnurrenberger, and Christian Tschudin. In-network live stream processing with named functions. In *16th International IFIP TC6 Networking Conference, Networking 2017*. IFIP Open Digital Library, 2017.

- [SMT19] Christopher Scherb, Claudio Marxer, and Christian Tschudin. Execution plans for serverless computing in information centric networking. In *Proceedings of the 1st ACM CoNEXT Workshop on Emerging in-Network Computing Paradigms*, pages 34–40. ACM, 2019.
- [SST14] Christopher Scherb, Manolis Sifalakis, and Christian Tschudin. *Computing the Distribution of Computations for Named Function Networking Using Name Based Routing*, University of Basel, 2014.
- [SST16] Christopher Scherb, Manolis Sifalakis, and Christian Tschudin. A packet rewriting core for information centric networking. In *2016 13th IEEE Annual Consumer Communications & Networking Conference (CCNC)*, pages 67–72. IEEE, 2016.
- [ST18] Christopher Scherb and Christian Tschudin. Smart execution strategy selection for multi tier execution in named function networking. In *IEEE ICC 2018*, 2018.
- [SLY⁺16] Hanbyul Seo, Ki-Dong Lee, Shinpei Yasukawa, Ying Peng, and Philippe Sartori. Lte evolution for vehicle-to-everything services. *IEEE communications magazine*, 54(6):22–28, 2016.
- [Ser18] Serverless. Serverless: use cases. <https://serverless.com/learn/use-cases/>, 2018. [Online; accessed 27-Nov-2018].
- [SCZ⁺16] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu. Edge computing: vision and challenges. *IEEE Internet of Things Journal*, 3(5):637–646, 2016. ISSN: 2327-4662.
- [SKS⁺14] Manolis Sifalakis, Basil Kohler, Christopher Scherb, and Christian Tschudin. An information centric network for computing the distribution of computations. In *Proceedings of the 1st ACM Conference on Information-Centric Networking*, pages 137–146. ACM, 2014.
- [SGF⁺10] Harald Sundmaeker, Patrick Guillemin, Peter Friess, and Sylvie Woelfflé. Vision and challenges for realising the internet of things. *Cluster of European Research Projects on the Internet of Things, European Commission*, 3(3):34–36, 2010.
- [TLM⁺19] Dominic Tarr, Erick Lavoie, Aljoscha Meyer, and Christian Tschudin. Secure scuttlebutt: an identity-centric protocol for subjective and decentralized applications. In *Proceedings of the 6th ACM Conference on Information-Centric Networking*, pages 1–11, 2019.

- [Tsc19] Christian Tschudin. A broadcast-only communication model based on replicated append-only logs. *ACM SIGCOMM Computer Communication Review*, 49(2):37–43, 2019.
- [TS⁺18] Christian Tschudin, Christopher Scherb, et al. Ccn lite: lightweight implementation of the content centric networking protocol, 2018.
- [TS14] Christian Tschudin and Manolis Sifalakis. Named functions and cached computations. In *Consumer Communications and Networking Conference (CCNC), 2014 IEEE 11th*, pages 851–857. IEEE, 2014.
- [TW16] Christian Tschudin and Christopher Wood. File-like icn collection (flic). *Internet Engineering Task Force, Internet-Draft draft-tschudin-icnrg-flic-00*, 2016.
- [Tur37] Alan M Turing. Computability and lambda definability. *The Journal of Symbolic Logic*, 2(4):153–163, 1937.
- [WSG⁺14] Md Whaiduzzaman, Mehdi Sookhak, Abdullah Gani, and Rajkumar Buyya. A survey on vehicular cloud computing. *Journal of Network and Computer Applications*, 40:325–344, 2014.
- [Wik18] Wikipedia. Lambda calculus. https://en.wikipedia.org/wiki/Lambda_calculus, 2018. [Online; accessed 06-Dec-2018].
- [Wik19] Wikipedia. Abstract syntax tree. https://en.wikipedia.org/wiki/Abstract_syntax_tree, 2019. [Online; accessed 03-Jan-2019].
- [ZWS⁺16] Haitao Zhang, Zhehao Wang, Christopher Scherb, Claudio Marxer, Jeff Burke, Lixia Zhang, and Christian Tschudin. Sharing mhealth data via named data networking. In *Proceedings of the 3rd ACM Conference on Information-Centric Networking*, pages 142–147. ACM, 2016.
- [ZAB⁺14] Lixia Zhang, Alexander Afanasyev, Jeffrey Burke, Van Jacobson, Patrick Crowley, Christos Papadopoulos, Lan Wang, Beichuan Zhang, et al. Named data networking. *ACM SIGCOMM Computer Communication Review*, 44(3):66–73, 2014.

Curriculum Vitae

Name Christopher Max Constantin Scherb
Alfred Holler Weg 9, 79540 Lörrach, Germany
Date of Birth 25 January 1989
Birthplace Lörrach, Germany
Citizenship Germany

Education

since 09.2014 PhD in Computer Science under the supervision of Prof. Dr. Christian Tschudin, Computer Network Group, University of Basel, Switzerland
2012 to 2014 Master in Computer Science at the University of Basel, Switzerland
Masterthesis: Computing the Distribution of Computations for Named Function Networking Using Name Based Routing
2009 to 2012 Bachelor in Computer Science at the University of Basel, Switzerland
Bachelorthesis: Hybrid Image Registration of Electron Microscopy Sequences
1999 to 2008 Abitur, Hebel Gymnasium Lörrach, Germany
1995 to 1999 Fridolin Grundschule Lörrach, Germany

Employment

- since 09.2014 Research and teaching assistant, Computer Network Group, University of Basel, Switzerland
- 08.2013 to 02.2014 Helping Researcher, Computer Networking, under the supervision of Prof. Dr. Christian Tschudin, Computer Network Group, University of Basel, Switzerland
- 03.2011 to 12.2011 Helping Researcher, Computer Vision, under the supervision of Prof. Dr. Thomas Vetter, Graphics and Vision Research Group, University of Basel, Switzerland
- 06.2010 to 07.2010 Summer Internship at Fraunhofer Institut, Ernst Mach Institut, Efringen Kirchen, Germany.
- 06.2009 to 08.2009 Summer Internship at Fraunhofer Institut, Ernst Mach Institut, Efringen Kirchen, Germany.

Publications

2012

- Christopher Scherb, Christoph Jud, and Thomas Vetter. *Hybrid Image Registration of Electron Microscopy Sequences*, Universität Basel, 2012.

2014

- Christopher Scherb, Manolis Sifalakis, and Christian Tschudin. *Computing the Distribution of Computations for Named Function Networking Using Name Based Routing*, University of Basel, 2014.
- Manolis Sifalakis, Basil Kohler, Christopher Scherb, et al. An information centric network for computing the distribution of computations. In *Proceedings of the 1st ACM Conference on Information-Centric Networking*, pages 137–146. ACM, 2014

2016

- Christopher Scherb, Manolis Sifalakis, and Christian Tschudin. A packet rewriting core for information centric networking. In *2016 13th IEEE Annual Consumer Communications & Networking Conference (CCNC)*, pages 67–72. IEEE, 2016

- Claudio Marxer, Christopher Scherb, and Christian Tschudin. Access-controlled in-network processing of named data. In *Proceedings of the 3rd ACM Conference on Information-Centric Networking*, pages 77–82. ACM, 2016
- Haitao Zhang, Zhehao Wang, Christopher Scherb, et al. Sharing mhealth data via named data networking. In *Proceedings of the 3rd ACM Conference on Information-Centric Networking*, pages 142–147. ACM, 2016

2017

- Christopher Scherb, Balázs Faludi, and Christian Tschudin. Execution state management in named function networking. In *IFIP Networking Conference (IFIP Networking) and Workshops, 2017*, pages 1–6. IEEE, 2017
- Christopher Scherb, Claudio Marxer, Urs Schnurrenberger, et al. In-network live stream processing with named functions. In *16th International IFIP TC6 Networking Conference, Networking 2017*. IFIP Open Digital Library, 2017

2018

- Christopher Scherb, Dennis Grewe, Marco Wagner, et al. Resolution strategies for networking the iot at the edge via named functions. In *2018 15th IEEE Annual Consumer Communications & Networking Conference (CCNC)*, pages 1–6. IEEE, 2018
- C Gundogan, C Scherb, T Schmidt, et al. Icn adaptation to lowpan networks (icn lowpan) draft-gundogan-icnrg-ccnlowpan-02, 2018
- Christopher Scherb and Christian Tschudin. Smart execution strategy selection for multi tier execution in named function networking. In *IEEE ICC 2018*, 2018
- Dennis Grewe, Claudio Marxer, Christopher Scherb, et al. A network stack for computation-centric vehicular networking. In *Proceedings of the 5th ACM Conference on Information-Centric Networking*, pages 208–209. ACM, 2018
- Christian Tschudin, Christopher Scherb, et al. Ccn lite: lightweight implementation of the content centric networking protocol, 2018

2019

- Christopher Scherb, Claudio Marxer, and Christian Tschudin. Execution plans for serverless computing in information centric networking. In *Proceedings of the 1st ACM CoNEXT Workshop on Emerging in-Network Computing Paradigms*, pages 34–40. ACM, 2019
- Christopher Scherb, Samuel Emde, Claudio Marxer, et al. Data upload in mobile edge computing over icn. In *Proc. of the IEEE Globecom 2019*. IEEE, 2019